Universidade Federal de Campina Grande Centro de Engenharia Elétrica e Informática Coordenação de Pós-Graduação em Informática

Isolamento Automático de Falhas em Sistemas

Wagner de Souza Porto

Campina Grande - PB
Setembro de 2009

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Isolamento Automático de Falhas em Sistemas

Wagner de Souza Porto

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, Campus I como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciência da Computação (MSc).

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

Jacques Philippe Sauvé

(Orientador)

Campina Grande - PB

Setembro de 2009

FICHA CATALOGRÁFICA

P853i Porto, Wagner de Souza
Isolamento automático de falhas em sistemas / Wagner de Souza Porto. ─ Campina Grande, 2009.
196 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Referências.
Orientador: Prof. Dr. Jacques Philippe Sauvé.

1. Diagnóstico de Falhas 2. Isolamento de Falhas 3. Propagação de Falhas I. Título.

CDU 004.41(043)

Resumo

Este trabalho apresenta o Auto-FDI (***Auto****matic* ***F****ault* ***D****etection and* ***I****solation*), uma ferramenta de detecção e isolamento de falhas em sistemas. A ferramenta usa o conceito de redundância analítica, onde sinais obtidos do sistema (possivelmente com falha) são comparados com sinais esperados, obtidos de um modelo. O isolamento de falhas emprega uma técnica desenvolvida neste trabalho, chamada *isolamento automático*. A técnica usa uma abordagem baseada em grafos que considera a propagação de falhas e a falta de informação sobre determinados componentes do sistema. Falhas são localizadas de forma mais precisa possível, dado o nível de detalhe do modelo. No escopo deste trabalho foi abordado todo o processo de especificação, projeto, implementação e validação da ferramenta, utilizada como prova de conceito para a técnica desenvolvida. A validação da ferramenta foi feita através da realização de um estudo de caso por potenciais usuários, o que permitiu demonstrar a aplicabilidade da ferramenta e a da técnica desenvolvida.

Palavras-Chaves: ferramenta, detecção de falhas, diagnóstico de falhas, isolamento de falhas, propagação de falhas, modelo de propagação de falhas, grafos.

Abstract

This work presents Auto-FDI (***Auto****matic* ***F****ault* ***D****etection and* ***I****solation*), a software tool for detection and diagnosis of faults in systems. The tool uses the analytical redundancy concept, where signals from the (possibly faulty) system are compared with expected signals from a model. The fault isolation employs a technique developed on this work, called *automatic isolation*. This technique uses a graph-based approach which considers the fault propagation and the lack of information about certain components of the system. Faults are pinpointed as accurately as possible given the level of detail in the model. In the scope of this work was addressed the whole process of specification, design, implementation and validation of the tool - used as proof of concept for the developed technique. The validation of the tool was made by conducting a case study for potential users, that has demonstrated the applicability of the tool and the technique developed.

Keywords: software tool, fault detection, fault diagnosis, fault isolation, fault propagation, fault propagation model, graph.

Agradecimentos

A Deus, que está presente em todos os momentos de minha vida;

A toda minha família, pelo apoio e incentivo constante durante essa jornada, especialmente aos meus pais;

A minha namorada, Adriana, pelo seu amor, carinho e compreensão;

Ao meu orientador, Jacques, por seus ensinamentos e pela confiança de que poderíamos realizar um bom trabalho;

Ao professor Jorge, que esteve tão presente ao longo do meu mestrado;

A Aninha e Vera, secretárias da COPIN, que com carinho e prontidão estiveram sempre dispostas a ajudar;

A toda equipe do projeto *Smart Analysis*, por contribuírem para que os objetivos deste trabalho fossem atingidos;

Aos meus amigos e amigas, pelo apoio e imensa torcida;

Enfim, obrigado a todos que contribuíram de maneira direta ou indireta, durante essa fase de minha vida.

Conteúdo

1 Introdução 1
1.1 Objetivo da dissertação 4
1.2 Estrutura da dissertação 5
2 Detecção e diagnóstico de falhas em sistemas 6
2.1 Introdução 6
2.2 Atributos desejáveis em um sistema de DDF 8
2.3 Detecção e diagnóstico de falhas baseados em modelos 10
2.3.1 A fase de detecção de falhas 11
2.3.2 A fase de diagnóstico de falhas 13
2.4 Modelagem do sistema 16
2.4.1 Modelos teóricos vs. modelos experimentais 17
3 Uma abordagem para o isolamento de falhas 19
3.1 Introdução 19
3.2 Modelo de propagação de falhas 20
3.3 Isolamento de falhas 23
3.4 O algoritmo de Isolamento Automático de Falhas 28
3.4.1 Redução do grafo 29
3.4.2 Computação de *Br* 31
3.4.3 Computação de *Ps* 32
4 Levantamento de requisitos 34
4.1 Descrição geral da ferramenta 34
4.2 Usuários da ferramenta 35
4.3 Requisitos 36
4.3.1 Requisitos Funcionais 37
4.3.2 Requisitos não funcionais 40
5 Projeto arquitetural 42
5.1 Requisitos arquiteturais 42
5.2 Decisões de projeto de arquitetura 43
5.3 Estrutura da arquitetura 57
5.3.1 Módulos principais 57
5.3.2 Interfaces do sistema 64

5.3.3 Interfaces dos Componentes Modelica 70
5.4 Comportamento Dinâmico 72
6 Projeto detalhado e aspectos de implementação 78
6.1 Interface com o usuário 78
6.1.1 *SimForge* 79
6.1.2 *Auto-FDI Editor* 80
6.1.3 *Auto-FDI Executor* 83
6.2 Bibliotecas de componentes Modelica 84
6.2.1 *Biblioteca Padrão Modelica* 84
6.2.2 *ObjectStab* 86
6.2.3 *ExtendedPetriNet* 87
6.2.4 *Diagnostics* 88
6.3 Organização do sistema 90
6.3.1 *O pacote framework* 92
6.3.2 O pacote *isolation* 93
6.3.3 O pacote *detection* 94
6.3.4 O pacote *modelicacommunication* 94
7 Verificação e Validação 96
7.1 Metodologia de testes 96
7.2 Validação da ferramenta 97
7.2.1 Satisfação dos requisitos 98
7.2.2 Avaliação de desempenho do isolamento automático de falhas 107
8 Conclusão e Sugestões para Trabalhos Futuros 109
8.1 Sugestões para Trabalhos Futuros 111
Apêndice A - Terminologia 113
Apêndice B - Ferramentas de apóio à detecção e diagnóstico de falhas 115
1.1 DIAG 115
1.2 AMandD 117
1.3 ModBuild 118
1.4 FDI-TOOLBOX 120
1.5 Um Framework para Auxílio na Criação de Diagnosticadores de Falhas em Sistemas 121
1.6 Quadro comparativo 121
Apêndice C – Manual da ferramenta 123

1.1 Visão Geral do Auto-FDI 123
1.2 Iniciando o Auto-FDI 125
1.3 Criação do Modelo 126
1.4 Criação do Modelo de Detecção de Falhas 143
1.5 Criação do ambiente de diagnóstico 150
1.6 Realização do diagnóstico e avaliação dos resultados 160
1.7 Simulação do processo de detecção e diagnóstico de falhas 163
Apêndice D – Cenários de Isolamento Automático de Falhas 168
Bibliografia 177

Lista de Símbolos e abreviações

ARRs - Analytical Redundancy Relations

CHESF - Companhia Hidro Elétrica do São Francisco

DCS – Decentralised Control System

DDF - Detecção e Diagnóstico de Falhas

DSS - Decision Support System

FDI - Fault Detection and Isolation

IFAC - International Federation of Automatic Control

LS – Least Squares

MIMO – Multiple Input Multiple Output

P&ID - Process and Instrumentation Diagram

SAFEPROCESS - Fault Detection, Supervision and Safety of Technical Process

SCADA - Supervisory Control and Data Acquisition

SIMO – Single Input Multiple Output

SISO – Single Input Single Output

Lista de Figuras

Figura 1: (baseada em [KKKC2004]) Sistema composto por atuadores, planta e sensores. 7

Figura 2: Falha aditiva vs. Falha multiplicativa 8

Figura 3: Classificação das falhas em relação à dependência no tempo 8

Figura 4 (retirada de [Pontes2008]): Estrutura de um sistema de detecção e diagnóstico de falhas baseados em modelos 11

Figura 5: Etapas da Detecção de Falha 11

Figura 6 (baseada em [Munoz1996]): Geração de resíduos 12

Figura 7 (baseada em [Isermann2006]): Unificação de sintomas no processo de detecção e diagnóstico de falhas 14

Figura 8: Métodos de isolamento e identificação de falhas 15

Figura 9: Uso de métodos de classificação para o diagnóstico de falhas 15

Figura 10: Classificação de sistemas de acordo com a quantidade de entradas e saídas 17

Figura 11 (retirada de [Rao1993]): Sistema de esfriamento de ácido nítrico. (a) Diagrama de bloco funcional. (b). Modelo de propagação de falhas. 20
Figura 12: Modelo de propagação de falhas para um sistema com componentes de múltipla saídas 21
Figura 13: Modelo de propagação de falha para um determinado sistema 22
Figura 14: Exemplo de um FPM para um determinado sistema 24
Figura 15: Exemplo de um FPM para um determinado sistema 25
Figura 16: Exemplo de FPM com propagação de falhas em um ciclo 25
Figura 17: Exemplo de FPM sem propagação de falhas no ciclo 26
Figura 18 (retirada de [CS2001]): FPM para um determinado sistema 27
Figura 19: Redução do grafo (Exemplo 1) 30
Figura 20: Redução do grafo (Exemplo 2) 30
Figura 21: Usuários da ferramenta 35
Figura 22: Arquitetura geral do Auto-FDI 58
Figura 23: Arquitetura: Interface com o usuário 59
Figura 24: Padrão *Model-View-Controller* 60
Figura 25: Arquitetura: Lógica de negócio 60
Figura 26: Arquitetura: Modelo de Detecção 62
Figura 27: Arquitetura: Modelo do Sistema 63
Figura 28: Arquitetura: Web Services 64
Figura 29: Mecanismo de conexão da linguagem Modelica 70
Figura 30: Tela inicial da ferramenta Auto-FDI 79
Figura 31: Tela principal do SimForge 80
Figura 32: Tela principal do Auto-FDI Editor 81
Figura 33: Auto-FDI Editor – Componentes de alarmes 82
Figura 34: Auto-FDI Editor – Diferença entre os componentes de alarme 83
Figura 35: Tela principal do Auto-FDI Executor 84
Figura 36: Pacotes da Biblioteca Padrão Modelica 85
Figura 37: Pacotes da biblioteca *ObjectStab* 86
Figura 38: Pacotes da biblioteca ExtendedPetriNet 88
Figura 39: Pacotes da biblioteca Diagnostics 88
Figura 40: Principais pacotes da ferramenta Auto-FDI 91
Figura 41: Diagrama de classes do pacote *isolation* 93
Figura 42: Diagrama de classes do pacote detection 94

Figura 43: Diagrama de classes do pacote modelicacommunication 95
Figura 44: Verificação de inconsistências no modelo 100
Figura 45: Modelo de Detecção para o sistema de excitação 103
Figura 46: Modelo com componentes de ligação 104
Figura 47: Detecção de falhas usando técnicas baseadas em modelo do sistema e modelo de sinais 105
Figura 48: Desempenho do isolamento automático para grafos densos 108
Figura 49: Tela inicial do Auto-FDI 125
Figura 50: Criando um projeto novo no SimForge 127
Figura 51: Criando um arquivo no *SimForge.* 129
Figura 52: Área de montagem do modelo do *SimForge.* 130
Figura 53: Ajustando o tamanho da área de montagem do modelo 130
Figura 54: Representação visual do modelo do sistema de excitação do tipo ST1A ... 131
Figura 55: Inserindo o componente PSSGain 131
Figura 56: Componente *PSSGain* na área de montagem visual do modelo. 132
Figura 57: Modelo parcial do sistema de excitação. 133
Figura 58: Conectando componentes no SimForge. 134
Figura 59: Modelo parcial do sistema de excitação 135
Figura 60: Código Modelica com a definição dos parâmetros. 137
Figura 61: Alterando o parâmetro do componente *Sum*. 138
Figura 62: Modelo parcial do sistema de excitação 139
Figura 63: Definindo a causalidade do conector “Verror_input”. 140
Figura 64: Modelo parcial do sistema de excitação 140
Figura 65: Ferramentas de desenho do SimForge 141
Figura 66: Ícone do modelo do sistema de excitação 141
Figura 67: Mensagem de verificação de erros no modelo – Sem erros 142
Figura 68: Opções para simulação do modelo 142
Figura 69: Verificação do resultado da simulação do modelo 143
Figura 70: Representação visual do modelo de detecção de falhas 144
Figura 71: Modelo de detecção de falhas – adição do modelo do sistema 145
Figura 72: Modelo de detecção de falhas – adição dos adaptadores 146
Figura 73: Definindo o arquivo de entrada para o adptador 147
Figura 74: Modelo de detecção de falhas – 148
Figura 75: Modelo de detecção de falhas finalizado 149

Figura 76: Mensagem de verificação de erros no modelo de detecção – Sem erros 149
Figura 77: Ambiente de diagnóstico para o sistema de excitação 151
Figura 78: Escolhendo o modelo do sistema e o modelo de detecção no Auto-FDI Editor 152
Figura 79: Auto-FDI Editor – Elementos da interface do editor. 152
Figura 80: Auto-FDI Editor - Escolhendo uma imagem no editor 153
Figura 81: Representação visual do sistema 154
Figura 82: representação visual x estrutura do modelo 154
Figura 83: Alarmes 155
Figura 84: Alarmes de sintoma x alarmes de componentes 156
Figura 85: Adicionando um alarme de componente 156
Figura 86: Auto-FDI Editor – Adicionando uma nova aba 157
Figura 87: Escolhendo as variáveis a serem plotadas no gráfico 157
Figura 88: Auto-FDI Editor – Adição de gráficos 158
Figura 89: Auto-FDI Editor – Aba com imagem da estrutura do modelo 159
Figura 90: Auto-FDI Editor – Lista de parâmetros 159
Figura 91: Auto-FDI Editor – Adicionando parâmetros em uma nova aba 160
Figura 92: Escolhendo o ambiente de diagnóstico 161
Figura 93: Auto-FDI Editor – Ambiente de diagnóstico aberto 162
Figura 94: Execução do Diagnóstico - Identificação de uma falha no Gerador 163
Figura 95: Modelo de Detecção para o Sistema de Excitação 165
Figura 96: Modelo do Sistema de Excitação com componentes de injeção de falha ... 166
Figura 97: Injeção de Falhas: (a) modelo sem componente de injeção de falha; (b) modelo com componente de injeção de falha 166

Lista de Tabelas

Tabela 1: Nomenclatura usada para descrever o Isolamento Automático de Falhas 28
Tabela 2: Etapas do algoritmo de Isolamento Automático de Falhas 29
Tabela 3: Descrição dos principais pacotes da ferramenta Auto-FDI 92
Tabela 4: Quadro comparativo das ferramentas 122
Tabela 5: Tipo dos componentes do modelo do sistema de excitação 133

Tabela 6: Parâmetros do modelo do sistema de excitção 136
Tabela 7: Parâmetros dos componentes 138

Lista de Códigos Fonte

Código 1: Algoritmo de redução do grafo 31
Código 2: Chamada ao algoritmo de redução do grafo 31
Código 3: Computação de *Br* 32
Código 4: Computação de *Ps* 33
Código 5: Conector da linguagem Modelica 70
Código 6: Trecho de código Modelica com definição dos valores dos parâmetros do modelo. 136

1 Introdução

Capítulo 1

É notório que, nos últimos anos, o avanço tecnológico tem impulsionado a automação dos processos industriais. Conseqüentemente, os sistemas que compõem tais processos desempenham um papel cada vez mais importante (sistema aqui pode ser entendido um conjunto de entidades interdependentes, que interagem para realizar uma determinada função). O funcionamento inadequado desses sistemas pode trazer grandes perdas econômicas, causar danos ao meio ambiente e até mesmo colocar vidas humanas em perigo. Portanto, é importante localizar eventuais falhas o mais rápido possível, permitindo que ações corretivas sejam tomadas.

Por outro lado, a complexidade dos sistemas envolvidos nos processos industriais e a enorme quantidade de informação a ser considerada dificultam o processo de supervisão e controle das organizações. Sistemas modernos são tipicamente constituídos de uma grande quantidade de subsistemas (componentes) interligados. Quando ocorrem falhas em determinados componentes, o efeito se propaga, afetando outros componentes do sistema. Além disso, é comum que em alguns componentes do sistema não existam sensores de falhas (ou alarmes). Com isso não é possível conhecer o estado destes componentes (com falha ou sem falha), o que agrava ainda mais o problema.

Em meio a essa dificuldade, os operadores dos centros de supervisão às vezes não percebem o surgimento de eventuais falhas ou cometem erros no diagnóstico das mesmas. Diante disso, surge a necessidade de uma solução computacional que permita a automação do processo de supervisão das organizações através da instalação de sistemas de detecção e diagnóstico de falhas (DDF).

O processo de *detecção e diagnóstico de falhas* (DDF) em sistemas, também chamado *análise de falhas*, consiste em três fases [Gertler1998]:

- **Detecção de falhas –** Indicação da presença de falha(s) e o instante de detecção.

- **Isolamento de falha -** Determinação da localização da(s) falha(s).
- **Identificação de falhas** - Determinação do tamanho e tipo da(s) falha(s).

Os sistemas de detecção e diagnóstico de falhas permitem detectar e diagnosticar falhas em sistemas de forma rápida e confiável e com isso aumentar a eficiência do processo de diagnóstico e manutenção das organizações. Como benefícios diretos, podemos destacar: o aumento da disponibilidade, confiabilidade e segurança dos sistemas e a diminuição dos custos de operação e manutenção.

No passado, a supervisão automática de sistemas era realizada, em sua maioria, através da verificação de limiares de algumas variáveis importantes do sistema [Isermann2006]. Alarmes eram disparados caso esses limiares fossem ultrapassados, sinalizando possíveis falhas no sistema.

Esse método simples pode ser útil em alguns casos onde os sistemas monitorados possuem estado constante e onde as variáveis monitoradas não variam dinamicamente junto com condições de operação do sistema (como, por exemplo, devido a um aumento na produção). Porém, essa técnica não permite um diagnóstico de falhas rápido e, na maioria dos casos, não é possível ter um diagnóstico de falhas detalhado [Isermann2006]. Além disso, em sistemas complexos, com muitas variáveis sendo monitoradas, pode ocorrer um fato conhecido como "avalanche de alarmes", onde vários alarmes são disparados em um curto intervalo de tempo.

Nos últimos anos muita pesquisa tem sido realizada na área e vários métodos avançados de detecção e diagnóstico de falhas em sistemas têm sido propostos. Em particular, o IFAC Symposium SAFEPROCESS (*Fault Detection, Supervision and Safety of Technical Process*) tem contribuído para a publicação de uma grande quantidade de pesquisa. Destaca-se também a contribuição de vários livros, entre eles [Isermann2006], [KKKC2004], [Gertler1998] e [SFP2002].

Alguns dos métodos de detecção e diagnóstico de falha baseiam-se em redundância física (ou de hardware) onde utilizam-se elementos de hardware redundantes (como por exemplo, sensores e atuadores), para medir ou controlar uma determinada variável do sistema. Desse modo, é possível verificar a consistência entre os elementos redundantes e com isso identificar a ocorrência de falhas nesses elementos. A principal desvantagem da redundância física consiste no elevado custo de aquisição e manutenção. Além disso, redundância física requer espaço extra para os

elementos redundantes, se tornando inviável em algumas situações onde o espaço físico é limitado. O uso dessa abordagem tem se restringido a sistemas de segurança crítica [Munoz1996].

Outra abordagem utilizada é o diagnóstico de falhas baseado em modelos (redundância analítica), que utiliza um modelo que descreve o comportamento real do sistema. Através da comparação entre o comportamento do sistema e o comportamento esperado (obtido do modelo) é possível identificar anormalidades no sistema.

Há um crescente interesse na teoria e aplicação de soluções de detecção e diagnóstico baseada em modelos. De acordo com ([Isermann2006], [Gertler1998], [SFP2002]), essa abordagem tem sido a mais utilizada. Isso ocorre principalmente devido ao reduzido custo, pois não há a necessidade da instalação de componentes de hardware extra, uma vez que o diagnóstico de falhas baseado em modelos é realizado via software em um computador. Além disso, a detecção e diagnóstico de falhas baseados em modelos apresenta uma teoria bem consolidada, com uma grande publicação de estudos na área.

Podemos encontrar na literatura vários trabalhos que descrevem a implementação e aplicação de ferramentas de apoio ao diagnóstico de falhas em sistemas (uma descrição e um quadro comparativo das principais ferramentas são apresentados no Apêndice B). Dentre os trabalhos existentes, podemos destacar: DIAG [KKKC2004], AMandD (Advanced and Monitoring Diagnostic Systems) [KKKC2004], FDI-ToolBox [DAS+2006], a toolbox for design of diagnosis systems [Frisk et al], e toolbox for model-based fault detection and isolation [TBR+1999].

Embora algumas dessas ferramentas tenham sido aplicadas com sucesso na indústria, constatou-se que elas não contemplam várias características e funcionalidades que consideramos bastante relevantes. Entre os principais problemas encontrados, destacamos:

- Necessitam de configuração e/ou implementação de algoritmos adicionais para realizar o diagnóstico em novos sistemas, ou seja, não realizam o isolamento automático de falhas com base apenas no modelo do sistema.
- São difíceis de estender, em alguns casos impossibilitando a adição de novos métodos de diagnóstico de falhas;

- Não permitem usar conjuntamente técnicas de detecção e diagnóstico de falhas baseadas em modelo de sistema e modelos de sinais;
- A maioria não permite integração com outras ferramentas de apóio à detecção e diagnóstico de falhas (como por exemplo, ferramentas especializadas em modelagem), nem expõe sua funcionalidade, permitindo, por exemplo, que o resultado do diagnóstico seja exibido em outros sistemas.
- Algumas não dão suporte à modelagem usando determinados tipos de modelos, como por exemplo, redes de Petri, modelos booleanos e modelos representados por máquina de estados finitos;
- Não permitem interligar submodelos de tipos diferentes para composição de novos modelos (por exemplo, interligar um modelo representado por equações diferenciais com um modelo representado por redes de Petri);
- Requer um conhecimento profundo de todo o processo de detecção e diagnóstico para a construção de uma solução de diagnóstico;
- Possuem fraco suporte à simulação do modelo e do processo de detecção de falhas e mecanismos para validação do modelo e da solução de diagnóstico criados;

Diante do exposto, surgiu a motivação para a criação de uma nova ferramenta de detecção e diagnóstico de falhas em sistemas. A ferramenta contemplará algumas das características e funcionalidades que não são tratadas pelas ferramentas existentes. O trabalho está inserido em um projeto de P&D financiado pela CHESF, intitulado "*Smart Analysis*".

1.1 Objetivo da dissertação

Esta dissertação tem como objetivo a concepção de uma ferramenta de detecção e isolamento de falhas de sistemas utilizando técnicas baseadas em modelos. A ferramenta, chamada Auto-FDI (***A****utomatic* ***F****ault* ***D****etection and* ***I****solation*) visa preencher algumas das lacunas deixadas pelas ferramentas de diagnóstico existentes. Associado a este objetivo, está o de desenvolver e implementar uma técnica de isolamento de falhas que considere a propagação de falhas e a falta de informação sobre o estado de determinados componentes do sistema.

1.2 Estrutura da dissertação

O restante desta dissertação está organizada como segue. O **Capítulo 2** apresenta a teoria relacionada à detecção e diagnóstico de falhas em sistemas, com ênfase em técnicas baseadas em modelos. Também são abordadas as principais técnicas de detecção e diagnóstico de falhas encontradas na literatura. No **Capítulo 3,** é introduzida uma técnica de isolamento de falhas desenvolvida neste trabalho. O levantamento de requisitos, o projeto da arquitetura e aspectos de implementação da ferramenta proposta são tratados nos capítulos **4, 5** e **6** respectivamente. O **Capítulo 7** apresenta como foi feita a verificação e validação da ferramenta. E por fim, no **Capítulo 8,** estão as considerações finais e sugestões para trabalhos futuros.

Além destes capítulos, este documento inclui quatro apêndices. O **Apêndice A** contém a terminologia adotada neste trabalho. O **Apêndice B** contém uma análise de algumas ferramentas de apóio à detecção e diagnóstico de falhas existentes na literatura. No **Apêndice C,** é apresentado um tutorial para realização de um estudo de caso usando a ferramenta desenvolvida. **No Apêndice D** são expostos alguns cenários de configuração de sistemas com o respectivo resultado do isolamento de falhas.

2 Detecção e diagnóstico de falhas em sistemas

Capítulo 2

Este capítulo tem como objetivo apresentar os conceitos necessários para melhor entendimento deste trabalho. Será apresentada a teoria relacionada à detecção e ao diagnóstico de falhas em sistemas com ênfase em técnicas baseadas em modelos. A terminologia adotada encontra-se no apêndice A.

2.1 Introdução

Uma falha pode ser definida como um desvio de pelo menos uma característica própria ou variável de um sistema [IB1997]. Essas falhas podem levar a um funcionamento inaceitável dos sistemas, podendo acarretar elevadas perdas econômicas, e, dependendo do processo, trazer conseqüências desastrosas para a sociedade e para o meio ambiente. Portanto, é importante identificar eventuais falhas o mais rápido possível para que ações corretivas sejam tomadas. Tais fatores justificam o crescente interesse por sistemas de apoio à detecção e ao diagnóstico de falhas.

O processo de **detecção e diagnóstico de falhas** (DDF), também chamado **análise de falhas**, consiste em três fases [Gertler1998]:

- **Detecção de falhas –** Indicação da presença de falha(s) e o instante de detecção.
- **Isolamento de falha -** Determinação da localização da(s) falha(s).
- **Identificação de falhas** - Determinação do tamanho e tipo da(s) falha(s).

Refere-se às fases de detecção e isolamento de falhas conjuntamente como *diagnóstico de falhas*. Na prática, porém, a fase de identificação de falhas, embora útil, pode não justificar o esforço necessário. Assim, a maioria dos sistemas de DDF inclui apenas duas fases: detecção e isolamento [Isermann2006]. De fato, na maioria dos casos, sistemas que realizem apenas a detecção e isolamento de falhas são suficientes.

Depois do processo de detecção e diagnóstico de falhas devem ser tomadas as devidas ações para a proteção do sistema. Proteção são mecanismos que visam a suprimir um comportamento potencialmente perigoso do sistema ou evitar as

conseqüências desse comportamento [IB1997]. A proteção pode ser realizada pelos controladores dos centros de supervisão e controle ou de forma automática, pelo próprio sistema de detecção e diagnóstico de falhas.

Os sistemas supervisionados são geralmente constituídos por atuadores, planta (elementos da instalação técnica) e sensores (Figura 1). *u* e *y* são, respectivamente, os sinais medidos de entrada e saída.

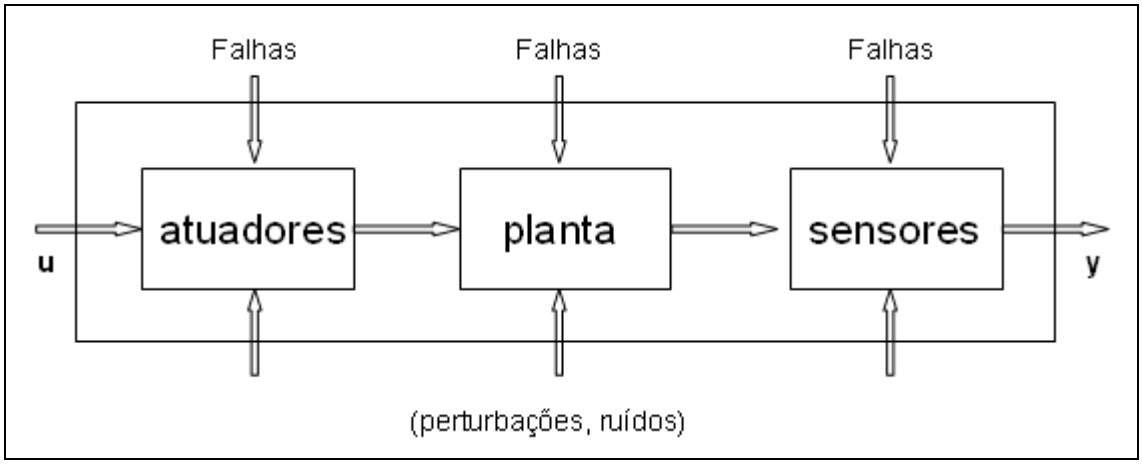


Figura 1: (baseada em [KKKC2004]) Sistema composto por atuadores, planta e sensores.

Conforme pode ser visto na Figura 1, as falhas podem ocorrer na planta ou nos dispositivos de controle e medida (atuadores e sensores) que integram o sistema. É importante também detectar e diagnosticar falhas em atuadores e sensores, pois estes equipamentos são bastante propensos a falhas.

Falhas podem ser vistas como entradas que afetam a saída do sistema. Além das falhas, os **distúrbios** e **ruídos** também afetam o sistema. Distúrbios são entradas adicionais desconhecidas, que atuam no sistema e que afetam a saída do sistema [Gertler1998]. A diferença entre falhas e distúrbios está no fato de que queremos detectar as falhas, enquanto que, desejamos ignorar os distúrbios e não sermos afetados por eles. Os ruídos, assim como os distúrbios, também são entradas adicionais desconhecidas, mas que possuem um comportamento aleatório e com média zero (se a média não for zero, são falhas ou distúrbios).

Um importante aspecto a ser considerado na detecção e diagnóstico de falhas são os tipos de falhas que podem afetar o sistema. De acordo com [Isermann2006], podemos distinguir entre os seguintes tipos de falhas:

- **Falhas Aditivas:** São falhas que causam uma mudança na saída do sistema através da adição delas mesmas. Essa mudança independe de outras entradas conhecidas (veja Figura 2-a).
- **Falhas Multiplicativas:** São falhas que causam mudanças em parâmetros do sistema. Essas falhas causam mudanças na saída que dependem também da magnitude de entradas conhecidas. São representadas pelo produto da falha f(t) por outra variável (veja Figura 2-b).

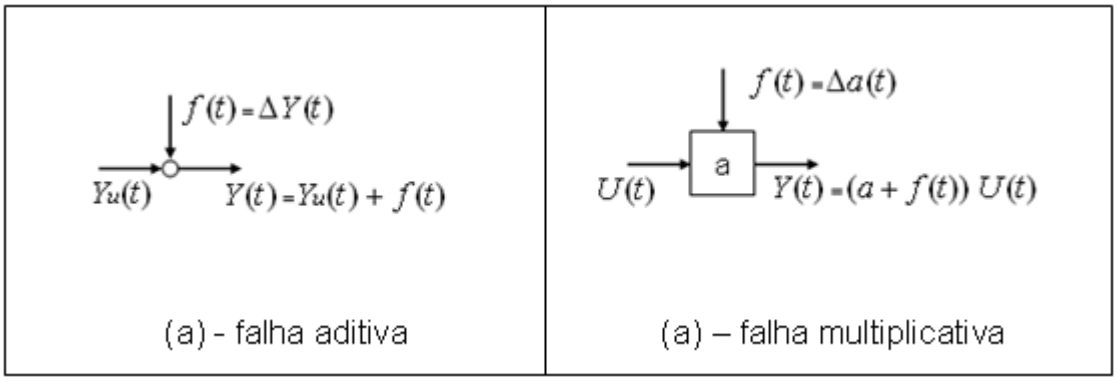


Figura 2: Falha aditiva vs. Falha multiplicativa

Em relação à dependência no tempo as falhas são classificadas de acordo com a Figura 3.

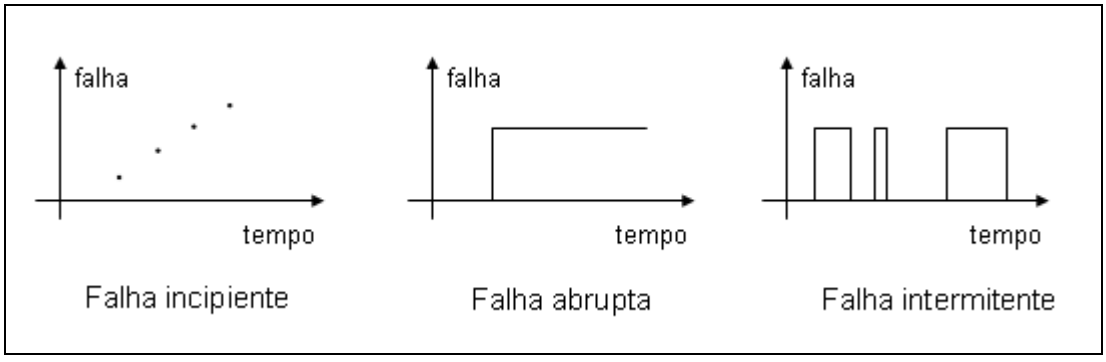


Figura 3: Classificação das falhas em relação à dependência no tempo

2.2 Atributos desejáveis em um sistema de DDF

Nesta seção apresentamos algumas características desejáveis em sistemas de detecção e diagnóstico de falhas. Idealmente, os sistemas de DDF deveriam possuir todos esses atributos; porém, esta é uma tarefa bastante difícil de ser alcançada. A lista de atributos apresentada aqui é útil para comparar soluções de DDF e direcionar o projeto de tais sistemas.

São atributos desejáveis em um sistema de diagnóstico de falhas [DV2000]:

- **Rapidez e precisão -** São atributos altamente desejáveis em sistemas de detecção e diagnóstico de falhas. Porém, esses atributos muitas vezes são conflitantes e, obviamente, existe um *tradeoff* entre eles.
- **Isolabilidade -** corresponde à habilidade em distinguir diferentes tipos de falhas. Deseja-se completude na isolabilidade de falhas, ou seja, o sistema deve ser capaz de distinguir todas as falhas possíveis no universo de falhas. Além disso, a corretude é um aspecto importante a ser considerado, de modo a evitar o surgimento de "alarmes falsos".
- **Robustez** - Deseja-se que o sistema seja capaz de diminuir o impacto causado por ruídos e incertezas, de modo que seu desempenho caia de forma gradual (ao invés de uma queda abruta). Para isso é importante a identificação de ruídos e incertezas que possam influenciar o sistema.
- **Identificação de novidade** (*Novelty Identifiability*) - Capacidade de diferenciar entre o funcionamento normal e anormal do sistema sendo monitorado, incluindo a identificação de falhas novas (desconhecidas). Deseja-se que essas falhas não passem despercebidas e também não sejam confundidas com falhas já conhecidas.
- **Identificação de múltiplas falhas -** A identificação de múltiplas falhas ocorrendo simultaneamente é um requisito altamente desejado em um sistema de DDF, porém é uma tarefa bastante difícil de ser alcançada. Modelar o efeito combinado de várias falhas não é uma tarefa fácil. Por outro lado, verificar a combinação de todas as falhas possíveis se torna computacionalmente inviável em sistemas complexos.
- **Facilidade de explicação** - Além de identificar a fonte das falhas, um sistema de DDF deveria prover explicações sobre o que originou as falhas e de como elas se propagaram até a situação atual. Essa informação permite que ações sejam tomadas para evitar que essas situações se repitam.
- **Adaptabilidade** - O sistema de DDF deve ser capaz de manter sua funcionalidade mesmo com mudanças no sistema sendo monitorado. Essas mudanças podem ser causadas por distúrbios ou devidas a outros fatores

(como por exemplo, uma mudança nas condições de operação devido a um aumento na produção).

2.3 Detecção e diagnóstico de falhas baseados em modelos

O processo de detecção e diagnóstico de falha baseados em modelos utiliza explicitamente um modelo do sistema ou de aspectos do sistema a ser monitorado. Através da comparação entre o comportamento do sistema e o comportamento esperado (especificado no modelo) é possível identificar anormalidades no sistema. Nesta abordagem existem dois tipos de modelos que podem ser usados: o **modelo de sinais** e o **modelo do sistema**.

O **modelo de sinais** considera o comportamento de oscilação ou de tempo cíclico de alguns sistemas, como por exemplo, máquinas de rotação e ruído acústico. Tanto os sinais periódicos, como os estocásticos, podem ser usados na detecção de falhas desde que mudanças nos sinais sejam causadas por falhas nos sistemas. O **modelo do sistema,** por sua vez, consiste em uma representação do sistema a ser modelado, contendo entradas, saídas, possíveis distúrbios e falhas.

A figura abaixo ilustra a estrutura básica de um sistema de detecção e diagnóstico de falhas baseados em modelos do sistema (Figura 4-a) e em modelo de sinais (Figura 4-b).

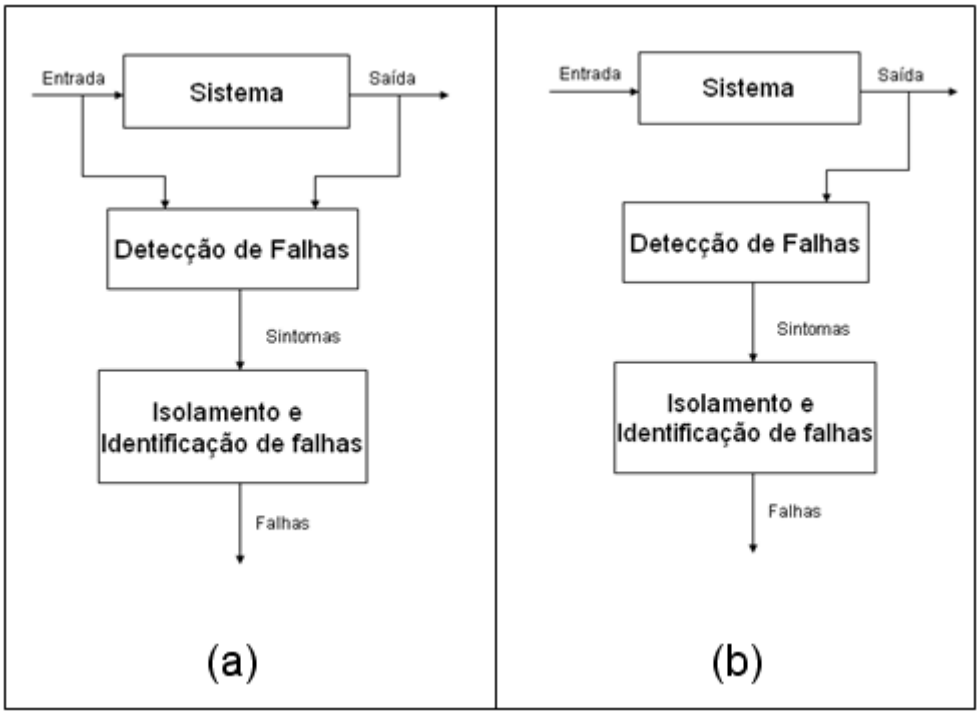


Figura 4 (retirada de [Pontes2008]): Estrutura de um sistema de detecção e diagnóstico de falhas baseados em modelos

A fase de **detecção de falhas** tem como resultado a geração de **sintomas**. Os sintomas indicam o estado do sistema (com falha ou sem falha). Após a detecção de falhas, segue o **diagnóstico de falhas**, que utiliza os sintomas para o isolamento e a identificação de falha através da aplicação de métodos de classificação ou de inferência [Isermann2004]. As seções seguintes descrevem estas duas fases em mais detalhes.

2.3.1 A fase de detecção de falhas

A fase de detecção de falhas, conforme ilustrado na Figura 5, envolve basicamente duas etapas: (1) **Geração de Resíduos** e (2) **Avaliação de Resíduos** [Gertler1998].

Figura 5: Etapas da Detecção de Falha

Na geração de resíduos é feita basicamente uma comparação entre os valores das variáveis do sistema com valores esperados. As diferenças entre essas variáveis, os resíduos, são indicadores da presença de falhas no sistema, baseado na diferença entre o

comportamento do sistema e o comportamento do modelo (comportamento esperado). A Figura 6 ilustra a geração de resíduos baseada no modelo de funcionamento normal da planta do sistema.

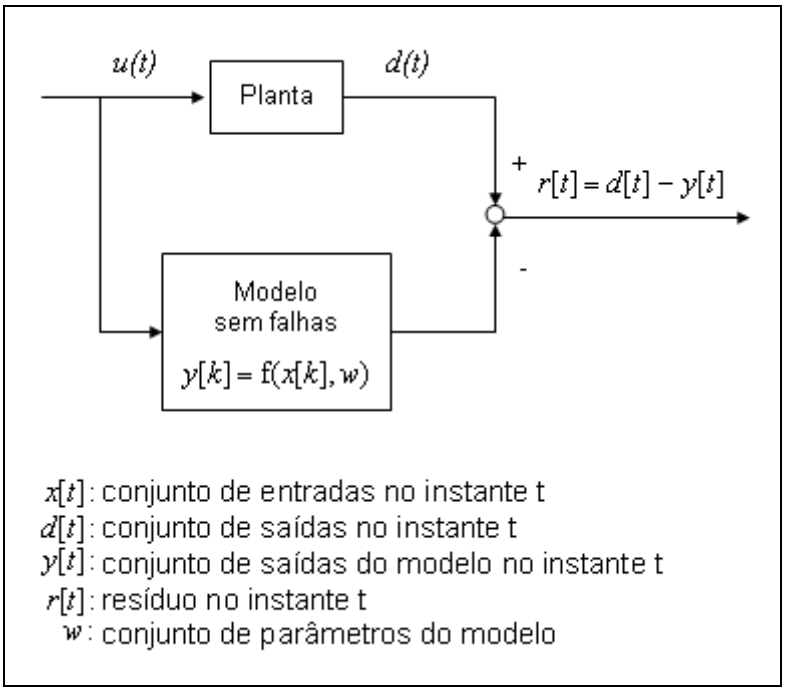


Figura 6 (baseada em [Munoz1996]): Geração de resíduos baseada no modelo de funcionamento normal da planta do sistema

Os resíduos são idealmente zero quando o sistema não possui falhas e um valor diferente de zero quando há falhas no sistema. Porém, devido à presença de ruídos, perturbações e erros de modelagem, os resíduos são geralmente diferentes de zero mesmo na ausência de falhas. Portanto é necessário haja uma avaliação dos resíduos para determinar se houve ou não falhas no sistema, tendo como resultado os sintomas. Os sintomas são a base para o diagnóstico de falhas. Eles podem ser valores binários [0, 1] ou, por exemplo, conjuntos *fuzzy*.

Um método bastante simples de avaliação de resíduos é a comparação do valor absoluto do resíduo com um limiar. O sintoma recebe o valor 1 (um) quando o limiar é excedido (indicando que uma falha foi detectada), ou 0 (zero) caso contrário:

$$S(r_j) = \begin{cases} 0 \ se \ |r_j| \leq K \\ 1 \ se \ |r_j| > K \end{cases}$$

Os limiares podem ser fixos, obtidos experimentalmente ou através de métodos estatísticos [Gertler1998]. Para diminuir a sensibilidade a distúrbios e ruídos, em vez de considerar o valor instantâneo dos resíduos, os sintomas poderiam ser gerados com base no valor médio dos N últimos valores de resíduos:

$$S(r_j) = \begin{cases} 0 \ se \ r_j(N) = \frac{1}{N}\left|\sum_{n=0}^{N-1} r_j, k-n\right| \leq K, \\ 1 \ se \ r_j(N) = \frac{1}{N}\left|\sum_{n=0}^{N-1} r_j, k-n\right| > K. \end{cases}$$

Diversos métodos de detecção de falhas são apresentados na literatura. Exemplos de métodos baseados em modelos do sistema são: estimativa de parâmetros, observadores de estado, estimativa de estado e equações de paridade e métodos que utilizam modelos de redes neurais e lógica *fuzzy*, entre outros. Exemplos de métodos baseados em modelos de sinais incluem: Filtros passa-faixa (*Bandpass filters*), análise de espectro (*Spectral analysis*) e estimativa de entropia máxima (*Maximum-entropy estimation*). Uma síntese de diversos métodos de detecção e diagnóstico de falhas e uma análise comparativa entre eles podem ser encontrados em [Isermann2006].

2.3.2 A fase de diagnóstico de falhas

O diagnóstico de falhas consiste na determinação do tipo, tamanho, localização e instante de detecção de uma falha [IB1997]. Esta fase utiliza os sintomas gerados na fase de detecção de falhas (**sintomas analíticos**) bem como sintomas obtidos através da observação do sistema por especialistas (**sintomas heurísticos**).

Os sintomas analíticos são valores quantitativos, enquanto que os sintomas heurísticos são geralmente valores qualitativos representados, por exemplo, através de expressões lingüísticas (barulho alto, muita fumaça, etc.). É importante, portanto, obter uma representação unificada desses sintomas para facilitar o processamento dos mesmos pelo sistema de diagnóstico. Esse processo, chamado de unificação de sintomas, é ilustrado na Figura 7.

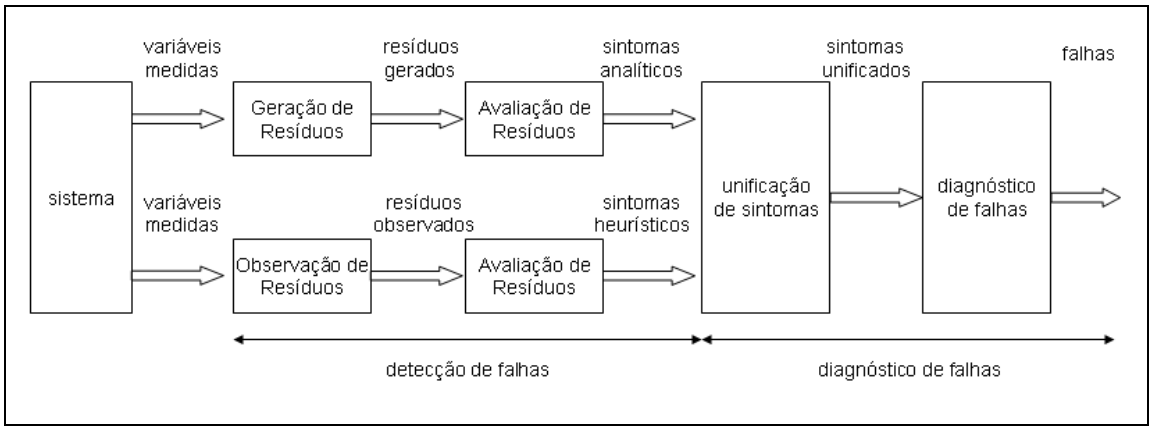


Figura 7 (baseada em [Isermann2006]): Unificação de sintomas no processo de detecção e diagnóstico de falhas

Existem vários mecanismos para se obter a unificação de sintomas. Uma possibilidade é a unificação através do uso de números de confiança e o tratamento probabilístico dos sintomas. A teoria por traz desse mecanismo é apresentada em [Freyumuth1991]. Outra possibilidade é a representação dos sintomas como funções de pertinência de conjuntos *fuzzy*, discutido em [IU1993]. Após a unificação dos sintomas, segue o diagnóstico de falhas propriamente dito.

Os métodos de diagnóstico de falhas podem ser divididos em dois grupos gerais: métodos de classificação e métodos de inferência. A Figura 8 mostra a relação de diversos métodos de diagnóstico de falhas que são abordados em [Isermann2006].

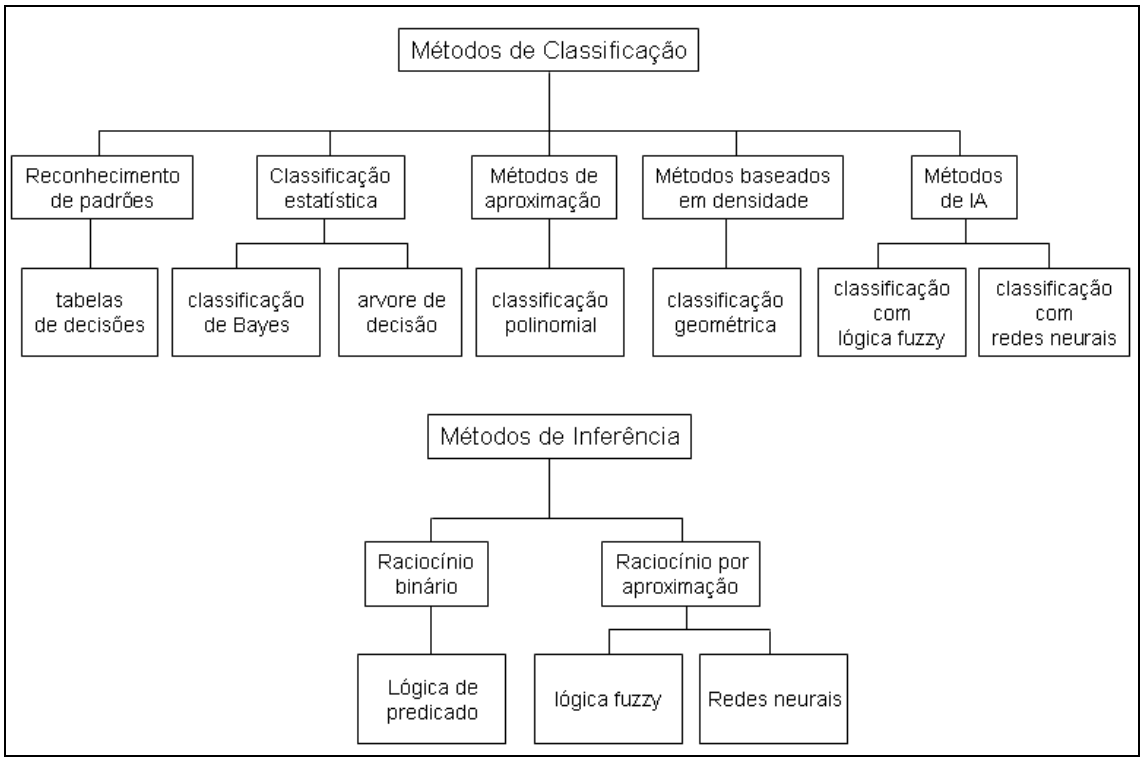


Figura 8: Métodos de isolamento e identificação de falhas

Os métodos de classificação são usados quando não se têm informações sobre as relações entre sintomas e falhas, ou seja, dado um conjunto de sintomas não se sabe, a priori, quais falhas geram esses sintomas. Conforme pode ser visto na Figura 9, esses métodos utilizam um conjunto sintomas de referência (*Sref*) para certas falhas no sistema (obtido experimentalmente através de treinamento ou aprendizagem). A partir da comparação entre o conjunto de referência e o conjunto de sintomas observados, é possível, através de classificação, determinar o conjunto de falhas (F) no sistema.

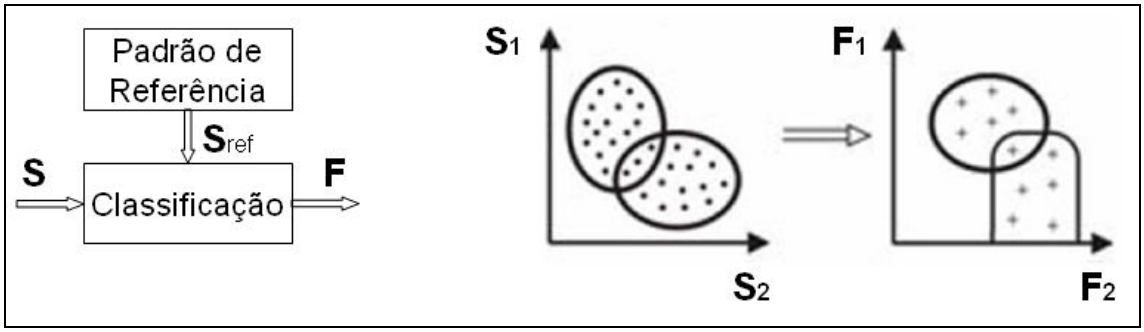


Figura 9: Uso de métodos de classificação para o diagnóstico de falhas

Por outro lado, quando se tem mais informação sobre as relações falha e sintomas, podem ser utilizados os métodos de inferência. Essas informações podem ser expressas, por exemplo, através de regras do tipo**: IF <sintomas> THEN <falhas>**. Em

[Isermann2006] é apresentado um estudo sobre o uso de métodos de inferência no diagnóstico de falhas, através do uso de árvores de decisão e lógica *fuzzy*.

Entre os métodos de diagnóstico de falhas, podemos citar também aqueles que utilizam um modelo do sistema representado por grafos [CS2001][Rao1993] [Rao1996][FPDM2001][DPRS1995]. Esses modelos, chamados freqüentemente de *modelos de propagação de falhas*, são excelentes para expressar a dinâmica de evolução de falhas em sistemas e tem se mostrado bastante útil no desenvolvimento de algoritmos de diagnóstico eficientes [FPDM2001].

Os métodos de diagnóstico baseadas em grafos tem despertado um grande interesse, devido a sua eficácia e grande aplicabilidade. Essa abordagem será discutida em detalhes no capítulo 3.

2.4 Modelagem do sistema

A construção do modelo é uma das questões mais importantes e mais difíceis quando empregamos a técnica de DDF baseada em modelos. Um "bom" modelo, que reproduza da maneira mais fiel possível o comportamento do sistema, pode melhorar significativamente o resultado da análise de falhas [SFP2002].

Para obter um "bom" modelo, é necessário conhecer bem as propriedades dos sistemas a serem modelados. De acordo com Gertler [Gertler1998], as propriedades mais importantes do sistema, são:

- **Estáticos versus dinâmicos:** Em sistemas estáticos, também chamados sistemas sem memória, as suas saídas, em qualquer instante, dependem apenas de suas entradas no mesmo instante. Em contraposição, em sistemas dinâmicos, as saídas em um determinado instante são influenciadas também por entradas ou saídas em outros instantes (passados ou futuros). Os sistemas estáticos são geralmente descritos através de equações algébricas enquanto que os sistemas dinâmicos são descritos através de equações diferenciais ou equações de diferença.
- **Lineares versus não-lineares:** Os sistemas lineares (estáticos ou dinâmicos) são aqueles onde o princípio da sobreposição é válido (a resposta para uma combinação de entradas é a mesma da soma das respostas individuas a cada uma das entradas). O princípio da sobreposição não se aplica aos sistemas

não-lineares. A maioria dos sistemas físicos são, de fato, não-lineares. Porém, muitas vezes, uma aproximação linear para esses sistemas, pode prover um resultado aceitável.

- **Modelos contínuos versus discretos:** Sistemas contínuos possuem os valores de suas variáveis definidos para qualquer instante de tempo enquanto que em sistemas discretos esses valores estão definidos apenas para alguns instantes de tempo (como por exemplo, a cada 10 segundos). Modelos contínuos são representados, por exemplo, através de equações diferenciais e modelos discretos através de equações de diferença ou suas variantes. A maioria dos sistemas físicos são inerentemente contínuos. Contudo, devido à natureza discreta dos computadores, esses sistemas são modelados de forma discreta.

Os sistemas também podem ser classificados de acordo com a quantidade de entrada e saídas, conforme ilustrado na Figura 10. Assim, temos os sistemas com uma única entrada e uma única saída (*single-input, single-output* – SISO), os sistemas com uma única entrada e múltiplas saídas (*single-input, multi-ouput* – SIMO), e os sistemas com múltiplas entradas e múltiplas saídas (*multi-input, multi-output* – MIMO).

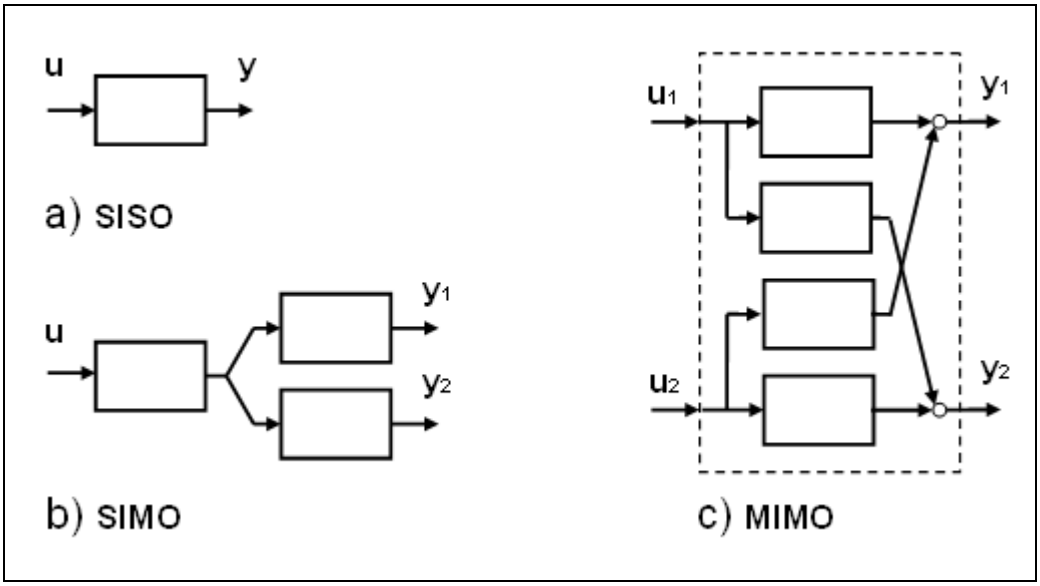


Figura 10: Classificação de sistemas de acordo com a quantidade de entradas e saídas

2.4.1 Modelos teóricos vs. modelos experimentais

Os modelos podem ser classificados em modelos teóricos ou modelos experimentais. Nos modelos teóricos, também chamados modelos caixa-branca, as

equações são formuladas através da análise das leis físicas que regem o sistema. Nesse método, cada elemento do sistema é analisado separadamente e suas equações são formuladas. O modelo é então representado pelo conjunto de equações desses elementos. Normalmente são feitas simplificações sobre o sistema através de linearização, redução da ordem da modelo, etc.

Os modelos experimentais, também chamados modelos caixa-preta são obtidos através de medição das variáveis de entrada e de saída do sistema. A partir da medição dessas variáveis e com a utilização de métodos de identificação, são obtidas as equações do modelo. Vários métodos de identificação de modelos são propostos na literatura, a exemplo do método de *Least Squares* (LS) [Isermann2006].

Durante a modelagem do sistema, uma técnica que pode ser utilizada para encontrar erros nos modelos é fazer a comparação entre o modelo teórico e modelo experimental do mesmo sistema. Se esses dois modelos apresentarem diferenças significativas nos resultados, significa que houve erros durante o processo de modelagem.

Um estudo detalhado sobre a modelagem do sistema para aplicação em diagnóstico de falhas pode ser encontrado em [SFP2002].

3 Uma abordagem para o isolamento de falhas

Capítulo 3

Neste capítulo introduzimos o Isolamento Automático de Falhas, uma abordagem de isolamento de falhas em sistemas baseada em grafo. No decorrer do capítulo será discutida a fundamentação teórica da técnica, fazendo um comparativo com técnicas existentes na literatura. Por fim, serão discutidos alguns aspectos de sua implementação.

3.1 Introdução

Sistemas industriais modernos estão cada vez maiores e mais complexos, o que torna o isolamento de falhas uma tarefa bastante difícil. Como já dissemos, tais sistemas são tipicamente constituídos de uma grande quantidade de subsistemas (dezenas ou centenas) interligados através de relações de conexões em série, paralelo, *feedback*, etc. Por subsistema (ou componente) entende-se como uma unidade funcional do sistema que possui uma função específica [GSHC2006].

Quando ocorrem falhas em componentes do sistema, o efeito pode se propagar para outros componentes, dificultando a localização da origem das falhas. Além disso, um outro fator agravante no isolamento de falhas é fato de que, geralmente, certos componentes do sistema não são observados, ou seja, não possuem sensores (ou alarmes) e, portanto, não é possível conhecer o estado destes componentes (com falhas ou sem falhas).

Diante do exposto, caracteriza-se a necessidade de uma abordagem de isolamento de falhas que considere tanto a propagação de falhas no sistema quanto a falta de informação sobre o estado de certos componentes.

Várias técnicas têm sido propostas na literatura para a solução de diversos problemas de isolamento de falhas – em [SS2004] é feita uma análise das principais técnicas usadas nos últimos anos. Entre elas, as técnicas baseadas em grafo têm despertado um grande interesse, devido a sua eficácia e grande aplicabilidade. Técnicas de isolamento de falhas baseadas em grafo utilizam um modelo chamado

freqüentemente de **modelo de propagação de falha** (*Fault Propagation Model* – FPM, em inglês). Esse modelo será descrito na seção seguinte.

3.2 Modelo de propagação de falhas

Um exemplo de um FPM bastante usado é o apresentado em [Rao1993], onde os vértices do grafo representam os componentes do sistema e as arestas representam a propagação de falhas entre esses componentes. Alguns dos vértices são equipados com alarmes, que disparam quando ocorre uma anormalidade no componente.

A Figura 11-(a), mostra um sistema de esfriamento de ácido nítrico. O FPM do sistema é mostrado na Figura 11-(b). Por exemplo, através do FPM podemos inferir que quando ocorre uma falha na válvula *V2* (um entupimento, por exemplo), pode haver uma alteração do ácido nítrico na saída do trocador de temperatura; uma falha no trocador de temperatura (vazamento de água no cano, por exemplo) também pode alterar a temperatura do ácido nítrico.

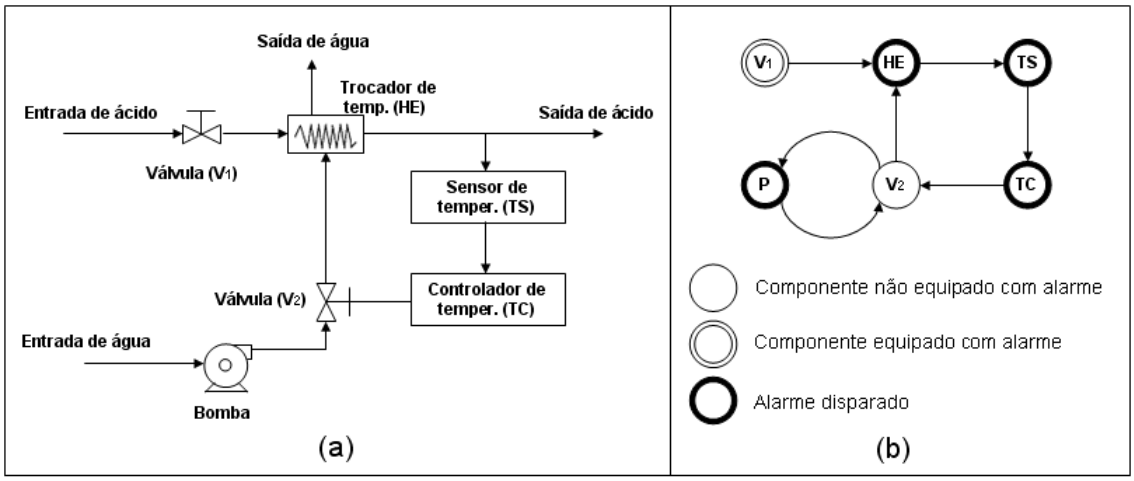


Figura 11 (retirada de [Rao1993]): Sistema de esfriamento de ácido nítrico. (a) Diagrama de bloco funcional. (b). Modelo de propagação de falhas.

Conforme ressaltado em [Rao1996] (do mesmo autor), esse modelo apresenta as seguintes propriedades: (a) uma falha em um componente se propaga para todos os seus vizinhos. (b) uma falha propagada em um vértice irá ativar o seu alarme.

Apesar do modelo descrito acima ter sido vastamente usado nos mais diversos contextos, é bem certo que ele não é válido para todos os casos, pois nem todos os sistemas satisfazem as propriedades (a) e (b) acima citadas. Um exemplo são sistemas que contêm componentes de múltiplas saídas (componentes do tipo SIMO ou MIMO –

veja seção 2.4), conforme o ilustrado na Figura 12-(a). Nesses tipos de componentes é possível que falhas afetem apenas uma de suas saídas enquanto que outras não, violando assim a propriedade (a). No sistema da Figura 12-(a), por exemplo, pode acontecer de uma falha no componente *A* se propagar para o componente *B* e não se propagar para os componentes *C* e *D*.

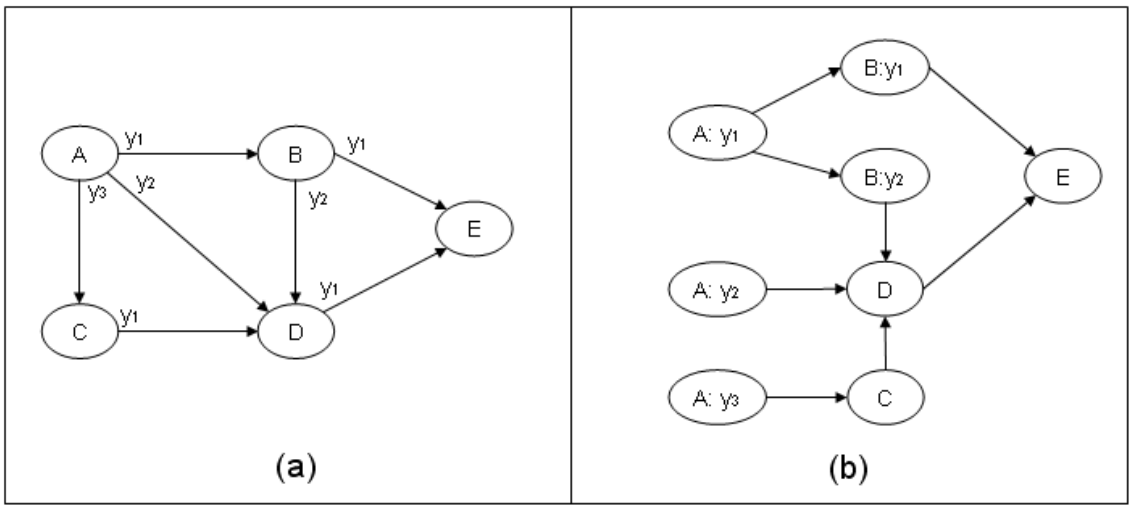


Figura 12: Modelo de propagação de falhas para um sistema com componentes de múltipla saídas

Embora esse FPM não contemple esses tipos de sistema, ainda assim é possível obter um modelo de propagação equivalente, conforme ilustrado na Figura 12-(b). O modelo é obtido criando um vértice para cada uma das saídas dos componentes de múltipla saída e conectados os vértices adequadamente. Porém, é visível que o modelo obtido é bem menos intuitivo, pois não tem um mapeamento direto entre os vértices do grafo e os componentes do sistema.

Por sua vez, temos que à propriedade "*uma falha propagada em um vértice irá ativar o seu alarme*", também nem sempre é válida. De fato, não é incomum que o alarme de um componente permaneça desativado mesmo que receba sinais de entrada de um componente com falha (geralmente quando a diferença no valor esperado de sinal está abaixo de um determinado limiar).

Sendo assim, usamos neste trabalho um modelo de propagação de falhas ligeiramente diferente do modelo apresentado, definido a seguir. O modelo contém as relações de dependências entre os componentes do sistema e também os sintomas gerados na fase de detecção de falha.

Definição 1: Seja $FPM = (V, E, f)$ um grafo dirigido representando o sistema, onde:

- $V = \{s_1, s_2, s_3, ..., s_n\}$ é o conjunto de vértices representando os componentes do sistema.
- $E \subseteq V \, x \, V$ é o conjunto de arestas do tipo e=(si, sj), onde si e sj são vértices de V. Se $(si, sj) \in E$ implica que os sistemas si e sj estão interligados (relação de dependência entre si e sj) e uma falha pode se propagar do componente si para o componente sj.
- $f : E \rightarrow \{OK, F, Ind\}$ é uma função de marcação de arestas. Essa função atribui uma marcação para as arestas do grafo com base nos sintomas gerados na fase de detecção de falhas. A função é definida por:

$f(si, sj) =$ {
- OK, se a saída do componente si apresenta o comportamento esperado.
- F, se a saída do componente si não apresenta o comportamento esperado (sintoma de falha).
- *Ind*, se não foi possível observar o estado da saída do componente si.

A Figura 13 ilustra o FPM em questão para o mesmo sistema da Figura 12-(a). Veja que o modelo é bastante intuitivo, com uma correspondência quase que direta com o sistema real (diferenciando apenas pela adição dos sintomas nas arestas e a inclusão de vértices adicionais representando as folhas do grafo). No exemplo, o componente *A* do sistema contém falhas.

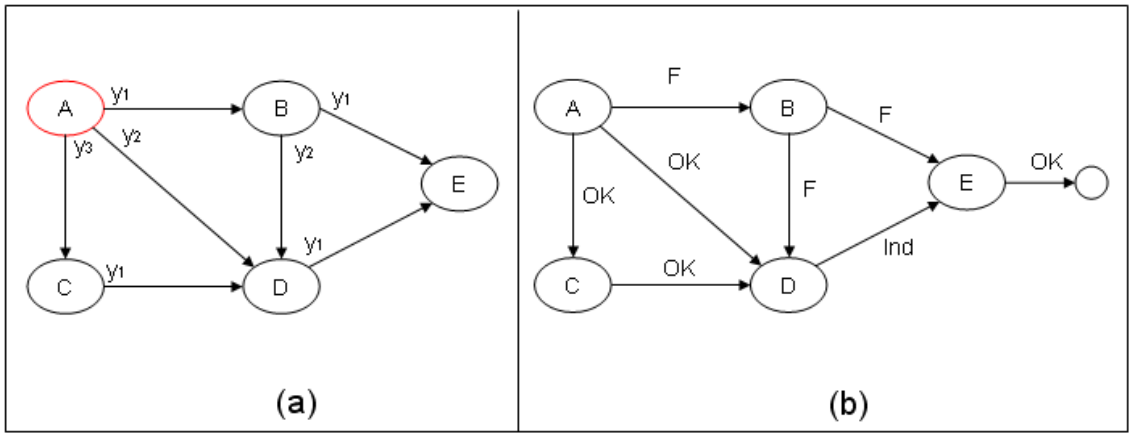


Figura 13: Modelo de propagação de falha para um determinado sistema

Com relação ao modelo, há várias observações a serem feitas. Primeiro, observe que a falha no componente *A* afetou apenas a variável y_1; as variáveis y_2 e y_3 não foram afetadas. Observe também que a variável y_1 do componente *D* não é observável (estado igual a "*Ind*"). E finalmente, veja que a falha se propagou pelo componente *B*, mas não se propagou pelo componente *E*. Como já discutimos anteriormente, o fato de uma falha se propagar por determinados componentes do sistema e não se propagar por outros é uma característica bastante comum em sistemas reais e, portanto, deve ser considerada.

3.3 Isolamento de falhas

O isolamento de falhas, conforme definido em [Gertler1998], consiste na localização da(s) falha(s) em um sistema, ou seja, tem como objetivo a identificação do(s) componente(s) do sistema com falha.

Em [Rao1993] é extensivamente abordado o problema do isolamento com uma única falha (*single fault diagnosis problem*), onde é considerada a hipótese de que não ocorre mais de uma falha no sistema ao mesmo tempo. No mesmo artigo é apresentada uma solução eficiente para este problema. Mais especificamente, é apresentado o algoritmo *FORWARD* e *BACKWARD*, com complexidade no pior caso de $O(n^2)$ e $O(kn+e)$ respectivamente, onde *n* é o número de vértices, *e* é o número de arestas e *k* é o número de vértices do grafo equipados com alarmes.

Em um outro trabalho ([CS2001]), é introduzido um algoritmo para o isolamento de falhas sob a suposição de que no máximo dois elementos do sistema podem apresentar falhas ao mesmo tempo (*double fault diagnosis problem*). O algoritmo, chamado D-FAULTS, com complexidade $O(n^3)$ no pior caso, onde *n* é o número de vértices do grafo.

No entanto, embora a hipótese pressuposta nesses estudos (única falha ou no máximo duas falhas) possa ser aplicada em alguns contextos, ela é irrealista para grande parte dos sistemas do mundo real. Devido à complexidade inerente aos sistemas industriais modernos, não são raros os casos em que vários componentes do sistema apresentam falha no mesmo instante de tempo.

Sendo assim, consideramos neste trabalho o isolamento em sistemas com múltiplas falhas. O problema consiste em determinar o conjunto de componentes

potencias fontes de falhas (*potential failure sources – Ps*, em inglês), levando em consideração a propagação de falhas e a falta de informação sobre determinados componentes do sistema (devido a ausência de sensores nestes componentes). Depois de determinar *Ps*, os componentes de *Ps* são testados fisicamente e as devidas ações são tomadas – reparando ou substituindo os componentes que realmente apresentarem falha.

Conforme mostrado em [Rao1993], o isolamento de falhas, quando considera um número arbitrário de componentes com falha, é NP-completo. Em [FPDM2001] é apresentado um algoritmo heurístico computacionalmente eficiente para este problema. A solução utiliza métodos de relaxamento de Langrangian e otimização de subgradiente. No entanto, por apresentar resultados probabilísticos, esta solução pode não ser aceitável em determinados contextos.

Para facilitar a compreensão do problema do isolamento de falhas, apresentamos alguns exemplos a seguir.

A Figura 14 é o FPM de um determinado sistema onde todos os componentes estão equipados com alarme e o componente *A* tem falhas. Observe que a falha se propagou pelos componentes *C* e *D*.

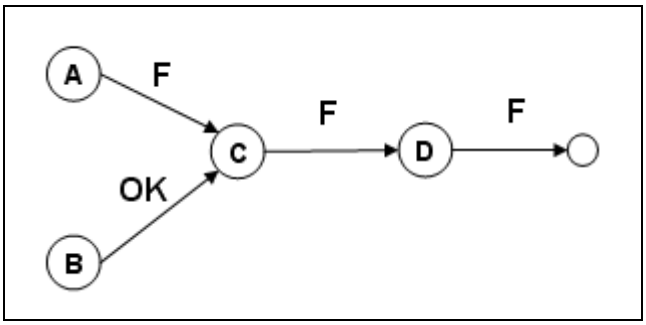


Figura 14: Exemplo de um FPM para um determinado sistema

Em uma situação como esta seria necessário, na pior das hipóteses, testar fisicamente todos os componentes do sistema – exceto o componente *B* que não apresentou sintomas de falhas, para identificar o componente com falha. Obviamente, o tempo e o custo para testar fisicamente um componente do sistema são geralmente bastante elevados. Ainda, na maioria das vezes, é necessário interromper o processo de produção para verificar o componente, causando grandes prejuízos econômicos.

Por este motivo, o isolamento de falhas visa reduzir o número de testes, identificando o conjunto de potenciais fontes de falhas. Nesse exemplo, é possível

inferir que o componente *A* é a fonte de falhas e que os sintomas de falhas apresentados pelos componentes *B* e *C* são devido à propagação da falha originada no componente *A*.

Considere agora um outro cenário, com FPM ilustrado na Figura 15. Observe que o componente *B* não é observável. Neste caso não é possível saber onde a falha originou no componente *A* ou no componente *C*. Assim, dizemos que os componentes *A* e *C* são potenciais fontes de falhas.

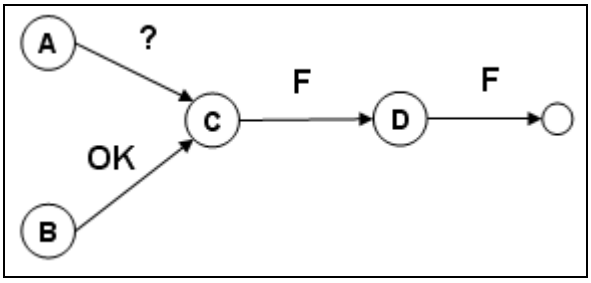


Figura 15: Exemplo de um FPM para um determinado sistema

Um aspecto importante a ser considerado no isolamento de falhas é a existência de ciclos de *feedback* nas relações de conexão do sistema. Quando uma falha ocorre em determinado componente do ciclo, a falha pode se propagar por todo o ciclo, impossibilitando a identificação da fonte da falha. Isso implica que o isolamento de falhas se limita a identificar o conjunto de componentes fortemente conectados com pelo menos um componente com falha. Este fenômeno é ilustrado na FPM da Figura 16, onde os componentes *A*, *B*, *C* e *D* são potenciais fontes de falha.

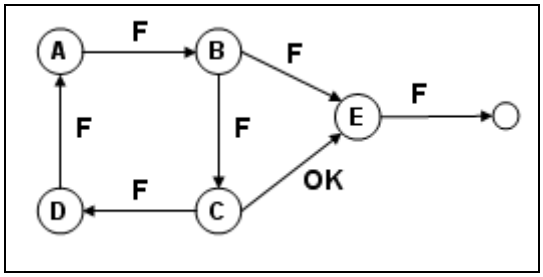


Figura 16: Exemplo de FPM com propagação de falhas em um ciclo

Por outro lado, em situações onde a falha não se propaga por todo o ciclo, é possível identificar a fonte das falhas. O FPM da Figura 17 ilustra um cenário para este caso, onde podemos inferir que o componente *A* e *C* são fontes de falhas.

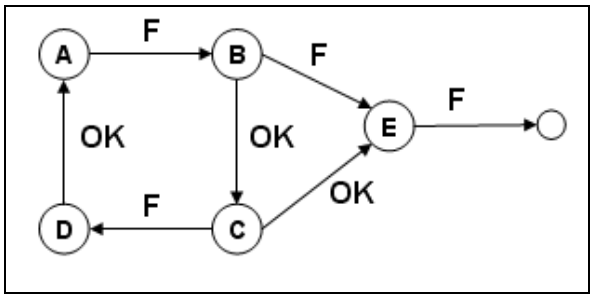


Figura 17: Exemplo de FPM sem propagação de falhas no ciclo

Em [CS2001] é apresentado uma solução para o isolamento de múltiplas falhas, chamada de diagnóstico operativo (*operative diagnosis*), que consiste em computar o conjunto de componentes S, de cardinalidade mínima, com pelo menos um componente com falha. Depois de identificado *S*, os componentes de *S* são então testados fisicamente e os componentes com falhas são substituídos ou reparados na fase de reparação. Esse procedimento é repetido até que todos os componentes com falhas do sistema sejam reparados.

A grande desvantagem do diagnóstico operativo é que ele não possibilita priorização ou paralelismo na reparação dos componentes com falhas. Além disso, o modelo de propagação utilizado (discutido na seção 3.2) possui as limitações já discutidas.

Em compensação, a solução concebida neste trabalho utiliza o FPM da Definição 1 (veja seção 3.2) e, diferentemente do diagnóstico operativo, consiste em determinar o conjunto que contém todos os componentes potencialmente fonte das falhas, definido como $Ps=\{F_1, F_2, ..., F_n\}$, onde cada $F_i \in Ps$ é o menor conjunto possível com pelo menos um componente com falha (e possivelmente componentes sem falhas). Chamamos isso de *isolamento automático de falhas*.

Para elucidar a diferença entre isolamento automático e o diagnóstico operativo, considere o cenário da Figura 18 (retirada de [CS2001]), em que os componentes c_2 e c_4 contêm falhas.

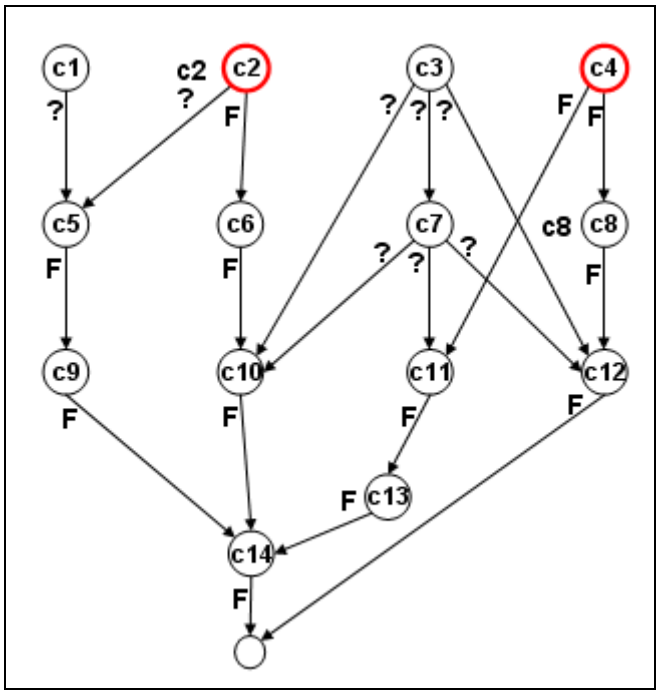


Figura 18 (retirada de [CS2001]): FPM para um determinado sistema

Com o diagnóstico operativo teríamos o seguinte:

1. Ao executar o algoritmo seria retornado o conjunto $S=\{c_1, c_2, c_5\}$ como sendo o menor conjunto com pelo menos um componente com falha.
2. Os componentes de *S* seriam testados fisicamente. O componente c_5 seria reparado (ou substituído) ao se constatar que ele contém falhas.
3. O algoritmo seria executado novamente, retornando dessa vez o conjunto $S=\{ c_4, c_{11}\}$.
4. Os componentes de *S* seriam testados. O componente c_4 seria reparado ao constatar que ele apresenta falha.
5. O algoritmo então seria finalizado, uma vez que o sistema não mais apresentaria falhas.

Através desse exemplo fica fácil perceber que o diagnóstico operativo não dá suporte à priorização ou paralelismo na reparação dos componentes com falhas. Por exemplo, não seria possível reparar o componente c_4 antes de reparar o componente c_2; ou reparar os dois componentes ao mesmo tempo. Essa característica pode impossibilitar a aplicação do diagnóstico operativo em vários contextos.

Já com o isolamento automático, seria retornado o conjunto *Ps* = { {c1, c2, c5 }, {c2, c3, c6, c7, c10}, {c3, c4, c7, c11}, {c3, c4, c7, c8, c12 }}. O apêndice D apresenta outros exemplos com várias configurações de modelos de propagação de falhas e o respectivo resultado do isolamento.

3.4 O algoritmo de Isolamento Automático de Falhas

Nesta seção apresentamos o algoritmo do Isolamento Automático de Falhas. O algoritmo recebe como entrada o FPM do sistema, que contém os sintomas dos componentes do sistema gerados na fase de detecção de falhas (veja seção 2.3.1.) e tem como resultado o conjunto de componentes potencias fontes de falhas, *Ps*.

No que segue, será adotada a seguinte nomenclatura:

Gr	Grafo reduzido do sistema (veja seção 0)
Vr	Conjunto de todos os vértices do grafo *Gr*.
Ar	Conjunto de todos os vértices do grafo *Gr* com marcação de falha (possui pelo menos uma aresta de saída com a marcação "*F*").
Cr	Conjunto de todos os vértices de *Gr* com marcação de falha e que são alcançáveis a partir de uma aresta com marcação de falha.
Br	Conjunto de todos os vértices do grafo *Gr* com marcação de falha e que não são alcançáveis a partir de outra aresta com marcação de falha. *Br = Ar – Cr*.
Ps	Conjunto de componentes potencias fontes de falhas (veja seção 3.3).
n	Número de vértices do grafo *Gr*.
e	Número de arestas do grafo *Gr*.
k_a	|*Ar*|, cardinalidade do conjunto *Ar*.
k_b	|Br|, cardinalidade do conjunto *Br*.

Tabela 1: Nomenclatura usada para descrever o Isolamento Automático de Falhas

O algoritmo, conforme mostra a Tabela 2, consiste de três etapas: Redução do Grafo, Computação de *Br* e Computação de *Ps*. A solução possui complexidade assintótica *O(kn + e)*, onde *k* é o número de componentes do sistema com sintomas de falhas e *n* e *e* são, respectivamente, o número de vértices e de arestas do grafo reduzido do sistema.

Etapas do Isolamento Automático	Complexidade de Tempo
1. Redução do Grafo	$O(n + e)$
2. Computação de *Br*	$O(k_a (n) + e)$
3. Computação de *Ps*	$O(k_b (n) + e)$

Tabela 2: Etapas do algoritmo de Isolamento Automático de Falhas

Nas seções seguintes cada uma das etapas do algoritmo é discutida em detalhes.

3.4.1 Redução do grafo

Conforme discutido anteriormente, quando ocorre uma falha em determinado componente de um ciclo do sistema e a falha se propaga por todos os componentes do ciclo, o isolamento de falhas se limita a identificar o conjunto de componentes fortemente conectados com pelo menos um componente com falha, pois não é possível identificar qual o componente que gerou a falha.

Em trabalhos correlatos ([Rao1993][Rao1996][CS2001]), utiliza-se um modelo reduzido do sistema obtido através da contração de todos os componentes fortemente conectados em um único vértice do grafo (pois consideram um modelo de propagação em que as falhas sempre se propagam). O resultado é um grafo acíclico, chamado grafo condensado do sistema (*system condensed graph*, em inglês).

Neste trabalho a redução do grafo consiste em obter um novo grafo *Gr* a partir do FPM, contraindo em um único vértice de *Gr* somente os componentes fortemente conectados em que todos os componentes do ciclo apresentam sintomas de falha ou possuem estado indeterminado. Os exemplos a seguir ilustram os grafo obtido através da redução do grafo, onde a região sombreada representa os componentes fortemente conectados a serem contraídos. No apêndice D podem ser encontrados vários outros exemplos.

Exemplo 1:

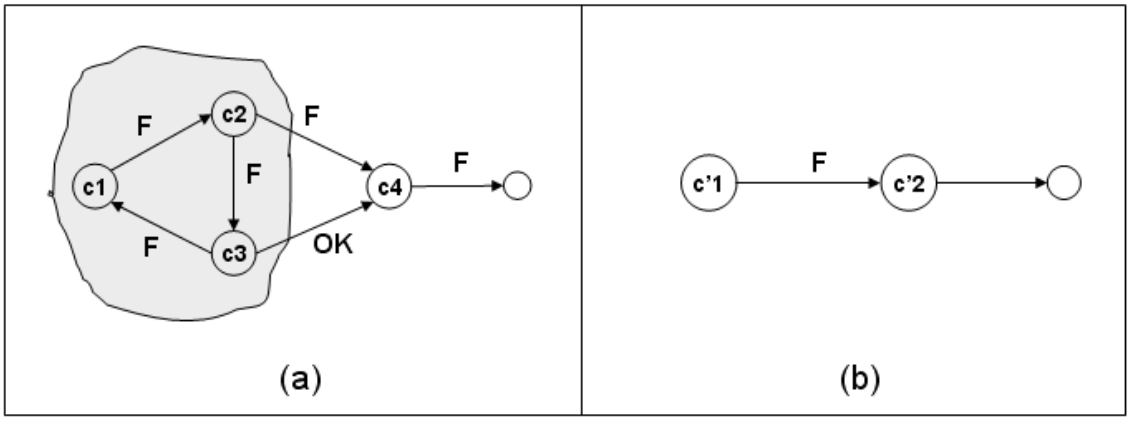


Figura 19: Redução do grafo (Exemplo 1)

Exemplo 2:

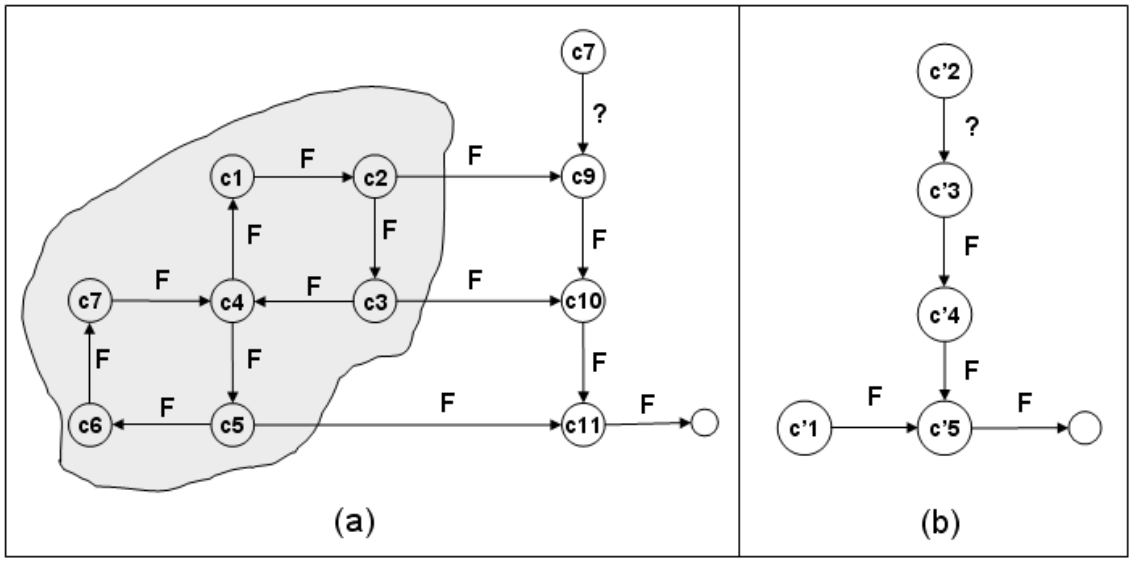


Figura 20: Redução do grafo (Exemplo 2)

A redução do grafo foi realizada usando o algoritmo de Tarjan [Tarjan1972] com uma ligeira alteração. A modificação no algoritmo original consistiu em verificar se existe uma aresta com marcação "*OK*" em alguma conexão dos componentes fortemente conectados, implicando que a falha não se propagou. O Código 1 mostra o pseudocódigo do algoritmo com a alteração em destaque. O algoritmo possui complexidade de tempo *O(n + e)*, onde *n* é o número de vértices e *e* o número de arestas do grafo [Tarjan1972].

```
GraphReduction (Graph v) {
  v.index = index                  // Set the depth index for v
  v.lowlink = index
```

```
  index = index + 1
  S.push(v)                          // Push v on the stack
  foreach e=(v, v') in E do          // Consider successors of v
    if (e.Tag != 'OK')
      break;
    if (v'.visited) then // Was successor v' visited?
      tarjan(v')                     // Recurse
      v.lowlink = min(v.lowlink, v'.lowlink)
    elseif (v' in S)then  // Is v' on the stack?
      v.lowlink = min(v.lowlink, v'.lowlink)

if (v.lowlink == v.index) then // Is v the root of an SCC?
    repeat
      v' = S.pop
      components[vertexIndex] <- components[vertexIndex] U v'
    until (v' == v)
}
```

Código 1: Algoritmo de redução do grafo[1]

Tendo em vista que o algoritmo de Tarjan encontra apenas os componentes fortemente conectados que são alcançáveis a partir do vértice inicial passado no parâmetro, para encontrar todos os componentes fortemente conectados é necessário fazer várias chamadas ao algoritmo, passando todos os vértices que não foram ainda visitados. Esta operação é ilustrada no trecho de código abaixo.

```
foreach v in V do
  if(!v.visited)
     v.visited=true
     GraphRedution(v)
```

Código 2: Chamada ao algoritmo de redução do grafo

3.4.2 Computação de *Br*

Esta etapa do algoritmo recebe o grafo reduzido *Gr* e consiste em determinar *Br*, o conjunto de todos os vértices do grafo *Gr* com marcação de falha e que não são alcançáveis a partir de outra aresta com marcação de falha. O pseudocódigo do algoritmo é exibido abaixo.

```
Cr = ∅
```

[1] Baseado em: Wikipedia. Tarjan's strongly connected components algorithm – The Free Encyclopedia, acessado em maio de 2008.

```
ComputeBr (Graph Gr ) {

        foreach v in Ar do
                if (!v.visited) then
                        DFS_Visit (Gr, v);
        Br = Ar - Cr
}

DFS_Visit (Graph g, Vertice v) {
        v.visited = true
        foreach u in v -> adj do
                e = g.GetEdge(u, v)
        if (e.Tag != 'OK' ) then
                Cr <- Cr ∪ {u}
                if(!u.visited)
                        DFS_Visit (g, u)
}
```

Código 3: Computação de *Br*

Em linhas gerais, o conjunto *Br* é obtido da seguinte maneira: Seja $Cr = \emptyset$. Faz-se um caminhamento em profundidade no grafo *Gr*, partindo de cada vértice $v \in Ar$, visitando as arestas com marcação "*F*" ou "*Ind*" (aresta com marcação "*OK*" é porque a falha não se propagou). Durante o caminhamento no grafo, todos os vértices visitados são adicionados em *Cr*, exceto o vértice *v*. O caminhamento em profundidade no grafo pode ser realizado em tempo $O(n + e)$ [DFS]. Logo, essa etapa do algoritmo é realizada em tempo $O(k_a(n) + e)$, onde $k_a = |Ar|$ e *n* e *e* são, respectivamente, o número de vértices e o número de arestas do grafo. Uma vez obtido *Cr*, o conjunto $Br = Ar - Cr$ é obtido em tempo $O(n)$. Assim, temos que *Br* é obtido em tempo $O(k_a (n) + e)$.

3.4.3 Computação de *Ps*

É onde de fato é realizado o isolamento de falhas, identificando o conjunto *Ps*. É realizado em duas etapas. Primeiro é computado o conjunto $Ps_0 = \{F_1, F_2, ..., Fk_b\}$, onde $k_b = |Br|$. Cada conjunto *Fi* é obtido através de um caminhamento em profundidade no Grafo *Gr* no sentido inverso das arestas, partindo de $vi \in Br$ e caminhando pelas arestas com marcação "*Ind*". Os vértices visitados são então adicionados no conjunto *Fi*. Tendo em vista que o caminhamento em profundidade no grafo é feito em tempo $O(n + e)$, onde *n* é o número de vértices e *e* é o número de arestas do grafo [DFS], essa primeira parte do algoritmo é realizada em tempo $O(k_b(n) + e)$, onde $k_b = |Br|$. A

segunda parte, que consiste em expandir os vértices condensados de *Ps,* é feito em tempo *O(n)*. Então, temos que o conjunto *Ps* é obtido em tempo $O(k_b(n) + e)$.

```
Ps = ∅
Cr = ∅
ComputePs (Graph Gr ) {

    foreach v in Br do
        Fi = ∅
        ComputeFi (v)

    //Expand the condensed vertices
    F = ∅
    foreach u in Fi do  //u is a condensed vertex
        F <- u.GetCondensedNodes
        Ps <- Ps  ∪ {F}

}
ComputeFi (Graph g, Vertice v) {

    Fi <- Fi  ∪ {v}
    foreach u in v -> inEdges do
        e = g.Edges[u, v]
        if (e.Tag == 'Ind' ) then
                ComputeFi (u)
}
```

Código 4: Computação de *Ps*

4 Levantamento de requisitos

Capítulo 4

Neste capítulo apresentamos a ferramenta proposta para a realização de detecção e isolamento de falhas em sistema, chamada Auto-FDI (***A**utomatic **F**ault **D**etection and **I**solation*). A ferramenta é resultado de um projeto de P&D financiado pela CHESF, intitulado "*Smart Analysis*". O capítulo está estruturado como segue. A seção 4.1 fornece uma visão geral da ferramenta, caracterizando o seu propósito geral. Na seção 4.2, são categorizados os potenciais usuários da ferramenta e em seguida, são apresentados os requisitos funcionais (seção 4.3.1) e não funcionais (seção 4.3.2) do sistema.

4.1 Descrição geral da ferramenta

O Auto-FDI é uma ferramenta que permite a detecção e isolamento de falhas em sistemas dos mais variados setores da indústria, em especial em sistemas de potência. A ferramenta utiliza técnicas de detecção baseadas em modelos de sistemas e/ou modelos de sinais para realização do diagnóstico. O isolamento de falhas é realizado de forma automática, através de métodos de isolamento de falhas baseados em grafos, sem necessidade de implementação de algoritmos especiais para diferentes sistemas. Falhas são identificadas de forma mais precisa possível, dado o nível de detalhe do modelo. O isolamento pode ser realizado para qualquer sistema para o qual se disponha de seu modelo estrutural.

Com o Auto-FDI é possível identificar falhas em sistemas de forma rápida e confiável, e com isso aumentar a eficiência do processo de diagnóstico e manutenção das organizações. Como benefícios diretos, podemos destacar o aumento da disponibilidade, confiabilidade e segurança dos sistemas e a diminuição dos custos de operação e manutenção.

Além de permitir o diagnóstico em sistemas reais, o Auto-FDI também pode ser usado para simulação, servindo como uma poderosa ferramenta de treinamento dos operadores de sistemas.

4.2 Usuários da ferramenta

Os usuários da ferramenta podem ser categorizados em três grupos distintos, de acordo com o seu papel: modelador, criador de diagnosticadores e o operador. Essa classificação é ilustrada no diagrama de contexto da Figura 21 e foi sugerida inicialmente por [Pontes2008], em um trabalho correlato.

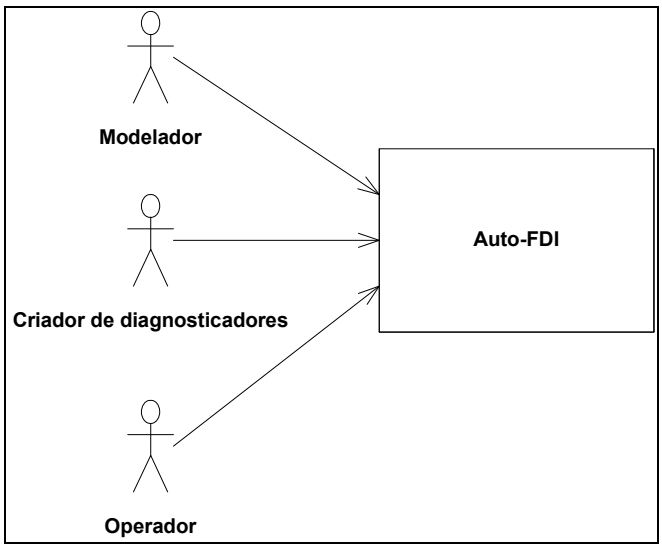


Figura 21: Usuários da ferramenta

Conforme já mencionamos, o Auto-FDI utiliza técnicas de detecção de falhas baseadas em modelos, para a realização do diagnóstico. Sendo assim, se faz necessário dispor de modelos que descrevam os sistemas a serem avaliados. O modelador é o usuário responsável por construir tais modelos. Este usuário, em geral um engenheiro do setor industrial onde a ferramenta está sendo aplicada, possui conhecimento sobre as características e propriedades físicas dos sistemas a serem modelados e geralmente tem bastante experiência na modelagem de sistemas.

O **criador de diagnosticadores** é o usuário responsável por montar um ambiente de diagnóstico adequado ao sistema a ser avaliado, ou seja, a interface que será usada pelo operador durante a realização do diagnóstico. O criador de diagnosticadores tem como propósito selecionar informações úteis que poderão ser configuradas e visualizadas durante a realização do diagnóstico. Este usuário possui conhecimento sobre o comportamento do sistema a ser avaliado, mas não precisa ter experiência na modelagem de sistemas.

O **operador** é o usuário que de fato realizará o diagnóstico de sistemas. Esses usuários geralmente são operadores dos centros de supervisão e controle da organização

onde a ferramenta está sendo usada. Esse tipo de usuário não precisa possuir avançado conhecimento técnico e geralmente reporta os resultados do diagnóstico para outra área da organização, para que, no caso de falhas, as devidas providências sejam tomadas.

Analisando os usuários da ferramenta, podemos perceber que existem diferenças entre os perfis (uns têm experiência com modelagem de sistemas, enquanto que outros não; uns possuem conhecimento sobre o sistema a ser avaliado e outros não). Essa característica foi um fator crucial durante o levantamento de vários requisitos do sistema, como por exemplo, o tipo de interface gráfica a ser disponibilizada para cada tipo de usuário.

4.3 Requisitos

Nesta seção apresentamos os requisitos funcionais e não funcionais da ferramenta. Segundo [Sommerville2004], os requisitos funcionais são descrições dos serviços fornecidos pelo sistema. Em alguns pode-se também estabelecer explicitamente o que o sistema não deve fazer. Já os requisitos não funcionais são restrições sobre os serviços ou funções oferecidos pelo sistema e atributos de qualidade dos mesmos.

Os requisitos da ferramenta foram obtidos a partir de conversas com os diversos *stakeholders* da ferramenta (cliente, usuários, desenvolvedores) e através da análise de ferramentas existentes (veja apêndice B). Alguns desses requisitos foram levantados por [Pontes2008], em um outro trabalho inserido no projeto *Smart Analysis*.

Os requisitos foram agrupados em conjuntos coerentes. Por exemplo, os requisitos funcionais foram classificados em *requisitos de criação de modelos*, *requisitos de criação de diagnosticadores*, etc.; os requisitos não funcionais em *requisitos de manutenibilidade, extensibilidade, custo*, etc. Cada requisitos possui um identificador único. O identificador inicia em [RF 1] ou [RNF 1] para requisitos funcionais e não funcionais respectivamente, e prossegue sendo incrementado ao passo que novos requisitos são adicionados.

É importante ressaltar que os requisitos apresentados aqui passaram por constante evolução à medida que se obteve uma maior compreensão do sistema por parte do cliente. Vários requisitos levantados inicialmente mudaram, enquanto que outros foram adicionados ou removidos. Um exemplo disso foi a remoção do requisito

"Identificação automática de modelos", presente na primeira versão da especificação do projeto:

> *Deve ser possível realizar a identificação de modelos funcionais de um sistema a partir dos dados monitorados, utilizando técnicas de identificação de modelos (como, por exemplos, técnicas de estimativa de parâmetros).*

Embora o requisito citado acima tenha sido removido, devido a uma mudança no escopo do projeto *Smart Analysis* por parte do cliente, consideramos tal requisito bastante importante em sistemas FDI e apresentamos como proposta para trabalhos futuros no capítulo 8.

O foco de pesquisa deste trabalho é o isolamento automático de falhas (veja requisito RF 13). Uma atenção especial também será dada aos requisitos **[Criação de modelos]**, **[Interligar modelos de tipos diferentes]**, **[Modelo de sistema e modelo de sinal]**, **[Simulação do diagnóstico]**, **[Arquitetura extensível]** e **[Exposição de funcionalidade]**, que foram os requisitos que de fato motivaram este trabalho.

4.3.1 Requisitos Funcionais

Os principais requisitos funcionais que nortearam o desenvolvimento da ferramenta são:

Requisitos de criação do modelo

RF 1. **[Criação de modelos]** Deve existir uma interface gráfica que permita criar modelos visualmente, através da composição de componentes de uma biblioteca de sub-modelos e funções. Deve ser possível criar os seguintes tipos de modelos:

- Modelos lógicos (booleano)
- Modelos contínuo estático, representado por equações algébricas;
- Modelo contínuo dinâmico, representado por equações diferenciais;
- Modelo de eventos discretos;
- Modelo probabilístico;
- Modelo representado por redes de Petri;
- Modelos representados por máquina de estados finitos;

RF 2. **[Interligar modelos de tipos diferentes]** Deve ser possível criar modelos interligando sub-modelos de tipos diferentes (como por exemplo, criar um modelo interligando um modelo lógico com um modelo contínuo).

RF 3. **[Salvamento/Leitura/Alteração/Exclusão de modelos**] A ferramenta deve permitir o salvamento, leitura, alteração e exclusão de modelos.

RF 4. **[Adição de modelos]** Deve ser possível adicionar facilmente na biblioteca de modelos da ferramenta, novos modelos criados pelo usuário;

RF 5. **[Verificação de consistência de modelos]** A ferramenta deve permitir que o usuário valide o modelo criado, ou seja, verifique se restrições estruturais estão sendo observadas, impedindo a criação de composições inválidas (por exemplo, conectando sinais de tipos incompatíveis).

Requisitos de execução do modelo

RF 6. **[Execução de modelos]** A ferramenta dever permitir a execução do modelo através da injeção de sinais de entrada e observação de sinais de saída. Para executar um modelo, deve ser possível fornecer valores para os parâmetros. A execução de modelos pode se apoiar em ferramentas especializadas (como por exemplo, *SimForge, Dymola*).

RF 7. **[Geração de sinais]** A ferramenta deve possuir componentes que gerem sinais. Tais sinais podem ser utilizados como entradas para modelos ou em qualquer outro ponto em que um sinal seja necessário para alimentar uma parte de um modelo. Deve ser possível gerar sinais por conteúdo especificado em arquivo, através de funções booleanas, funções contínuas algébricas e seqüências de eventos.

Requisitos de criação do ambiente de diagnóstico

RF 8. **[Criação do ambiente de diagnóstico]** A ferramenta deve possuir um editor para a criação do ambiente de diagnóstico – a interface que será usada pelo operador para realizar o diagnóstico de falhas no sistema. Deve ser possível criar uma interface personalizada de acordo com o sistema a ser avaliado. A interface criada poderá conter imagens representando o sistema a ser avaliado, figuras, links, gráficos, etc.;

RF 9. **[Gráficos]** O ambiente de diagnóstico deve permitir a geração de gráficos para visualização do processo de diagnóstico (valores de entradas e saídas do sistema, saída do modelo, resíduos, sintomas, falhas, etc.).

RF 10. **[Relatórios]** O ambiente de diagnóstico deve permitir a geração de relatórios com o resultado do diagnóstico.

RF 11. **[Salvamento/Leitura/Alteração/Exclusão do ambiente de diagnóstico]** A ferramenta deve permitir o salvamento, leitura, alteração e exclusão do ambiente de diagnóstico criado.

Requisitos de processo de diagnóstico

RF 12. **[Modelo de sistema e modelo de sinal]** Deve ser possível criar soluções de diagnóstico usando métodos de detecção de falha baseados: (a) em modelos de sistemas; (b) em modelos de sinais; (c) uma combinação de ambos.

RF 13. **[Realização do diagnóstico]** Deve ser possível realizar o diagnóstico através do ambiente de diagnóstico criado. Valores para parâmetros e para os sinais de entrada e saída oriundos de um sistema real devem ser informados. O diagnóstico deve indicar as possíveis falhas no sistema sendo avaliado. O isolamento de falhas deve ser realizado de forma automática, dado o modelo estrutural do sistema, sem necessidade de implementação de algoritmos especiais para diferentes sistemas. Falhas devem ser identificadas de forma mais precisa possível, dado o nível de detalhes do modelo. Deve ser possível realizar o diagnóstico em sistemas com até 1000 componentes em no máximo 10 segundos.

RF 14. **[Controle de execução do diagnóstico]** Durante a execução do diagnóstico, deve ser possível:

- Parar, pausar e reiniciar a execução a qualquer momento; e
- Controlar a velocidade do tempo da execução.

RF 15. **[Simulação do diagnóstico]** A ferramenta deve permitir a simulação do diagnóstico para fins de validação. Deve ser possível injetar falhas no sistema durante a simulação.

RF 16. **[Exposição de funcionalidade]** A ferramenta deve expor a funcionalidade relacionada ao processo de detecção e diagnóstico de falhas, permitindo e a integração com outras ferramentas.

4.3.2 Requisitos não funcionais

Os principais requisitos funcionais são:

Requisitos de Custo

RNF 1. **[Hardware]** A ferramenta deve ser executável em plataforma de hardware de baixo custo, tipo microcomputador, com processador de desempenho médio (tipo Pentium 4, 3 GHz com processador único), com memória de 1 GB.

RNF 2. **[Software básico]** A ferramenta deve ser executável em plataforma de software Windows.

Requisitos de Manutenibilidade

RNF 3. **[Arquitetura extensível]** A ferramenta deve possuir uma arquitetura que permita ser estendida facilmente com a adição de novos componentes funcionais. A adição de componentes poderá ser feita por terceiros, sem necessidade de recompilação de código.

RNF 4. **[Testes automáticos]** No sentido de manter a qualidade do software ao longo de sua evolução, toda a lógica de negócio deve ser testável automaticamente através de baterias de testes de funcionalidade (testes de aceitação). É desejável, mas não obrigatório, que a lógica de apresentação e a lógica de controle (interface entre a entrada/saída e a lógica de negócio) sejam testadas automaticamente.

Requisitos de Portabilidade

RNF 5. **[Linguagem/portabilidade]** A ferramenta deve ser desenvolvida na linguagem C# (Csharp) da plataforma .Net.

Requisitos de Interface e de Usabilidade

RNF 6. **[Usabilidade]** O sistema deverá ser amigável podendo ser facilmente utilizado após um treinamento básico.

RNF 7. **[Erros]** O sistema deve reportar qualquer erro com mensagens explicativas. Não se devem expor mensagens internas (do tipo "Null Pointer Exception") para o usuário.

RNF 8. **[Stand-alone]** A ferramenta deve ser do tipo stand-alone (monousuário).

Requisitos de Confiabilidade

RNF 9. **[Integridade]** O sistema deve ter um mínimo de controle na verificação da integridade dos seus dados internos de configuração.

Requisitos de Precisão

RNF 10. **[Precisão]** Dados numéricos devem manter a precisão natural associada ao tipo de informação sendo tratada.

Requisitos de Documentação

RNF 11. **[Documentação]** O sistema deve dispor de documentação de desenvolvimento.

RNF 12. **[Manual]** A ferramenta deve dispor de um manual do usuário.

Requisitos de Integração

RNF 13. **[Integração com ferramentas de modelagem]** Por força dos requisitos funcionais, o sistema poderá se integrar com ferramentas externas de modelagem. O sistema deverá oferecer toda sua funcionalidade de realização de diagnóstico através de uma interface exposta a sistemas externos, podendo estes utilizarem outra linguagem ou plataforma de execução.

5 Projeto arquitetural

Capítulo 5

Neste capítulo descrevemos a arquitetura da ferramenta em um nível abstrato de detalhamento. A forma escolhida para documentar a arquitetura do Auto-FDI foi baseada em [Sauve2007] e consiste de uma combinação de técnicas utilizadas por praticantes de engenharia de software:

- Identificação dos requisitos arquiteturalmente significativos para compor os *requisitos arquiteturais* (seção 5.1) [Bredemeyer-a]
- *Decisões arquiteturais* e comentários associados para apresentar a estrutura da arquitetura; assim, pode-se esclarecer a razão (o porquê) de cada decisão principal (seção 5.2) [Bredemeyer-b].
- *Visão lógica* da arquitetura, apresentada pelas seções "*Estrutura da arquitetura"* (seção 5.3) e "*Comportamento dinâmico"* (seção 5.4). Essas diferentes perspectivas ajudarão a melhor entender a arquitetura.

A arquitetura foi elaborada tendo em vista os princípios gerais de um bom projeto arquitetural, tais como abstração, modularização, alta coesão, fraco acoplamento, ocultação de informação, etc., o que propiciou eliminar os principais riscos no projeto e estabelecer uma arquitetura a partir da qual o sistema poderá evoluir.

5.1 Requisitos arquiteturais

Requisitos arquiteturais são aqueles que têm uma grande influência na determinação da arquitetura do sistema. São normalmente requisitos tecnicamente desafiadores, restritivos ou centrais para a finalidade do sistema. A identificação desses requisitos é de grande importância, pois eles são o foco principal do projeto da arquitetura.

Os requisitos arquiteturais do sistema Auto-FDI são listados a seguir. Usamos a abreviação introduzida no capítulo 4 e pedimos ao leitor que se dirija àquele capítulo para obter uma descrição completa dos requisitos.

Os requisitos arquiteturais são:

- Requisitos funcionais arquiteturais:
  - **[Criação de modelos]**
  - **[Interligar modelos de tipos diferentes]**
  - **[Adição de modelos]**
  - **[Verificação de consistência de modelos]**
  - **[Execução de modelos]**
  - **[Geração de sinais]**
  - **[Criação do ambiente de diagnóstico]**
  - **[Modelo de sistema e modelo de sinal]**
  - **[Realização do diagnóstico]**
  - **[Simulação do Diagnóstico]**
  - **[Exposição de funcionalidade]**

- Requisitos não funcionais arquiteturais:
  - **[Software básico]**
  - **[Arquitetura extensível]**
  - **[Testes automáticos]**
  - **[Stand-alone]**
  - **[Integração com ferramentas de modelagem]**

5.2 Decisões de projeto de arquitetura

De acordo com [Sommerville2004], o projeto de arquitetura é um processo criativo em que se tenta estabelecer uma organização de sistema (seus elementos e relações) de modo a satisfazer os requisitos funcionais e não funcionais do sistema. Durante esse processo, são tomadas várias decisões que afetam profundamente o sistema. Essas decisões são chamadas *decisões arquiteturais*.

Esta seção detalha as decisões arquiteturais que nortearam o projeto lógico do sistema. A documentação das decisões arquiteturais segue o *template* empregado em [Sauve2007]. Para cada decisão arquitetural, são indicados os *drivers* para a decisão, ou seja, os requisitos arquiteturais que levam à decisão descrita. Cada decisão é descrita mais detalhadamente, e suas vantagens e desvantagens são indicadas. Algumas decisões de "não fazer" também foram incluídas.

A documentação das decisões arquiteturais é essencialmente importante, pois possibilita o chamado *rastreamento de requisitos* na arquitetura, que é a possibilidade de relacionar os requisitos do sistema com os elementos (sub-sistemas) onde esses requisitos são implementados. Essa característica é importante porque facilita o entendimento e a manutenção do sistema representado pela arquitetura [Germoglio2008].

Vale ressaltar que, embora tenha se optado por usar tecnologias da plataforma .Net da Microsoft para a concretização da solução, não há implicação que apenas tais tecnologias sirvam ao propósito do sistema. Outras tecnologias poderão ser utilizadas com igual eficácia.

1	Decisão	**Usar uma arquitetura baseada em componentes**
	Drivers	Desacoplamento; RNF: **[Arquitetura extensível]**
	Descrição	A arquitetura do sistema deve ser projetada de forma que possa ser possível compor o sistema através da montagem de componentes. É muito importante observar que estamos usando aqui a palavra "componente" com uma semântica precisa, como definido por Wang e Kai Qian, em *Component-Oriented Programming* [WK2005]: "um pedaço de código auto-contido e auto-implantável com uma funcionalidade bem definida e que pode ser agregado com outros componentes através de sua interface". A forma de integrar componentes à solução final é de usar "*very late binding*" (usando, por exemplo, reflexão em C#).
	Vantagens	Uma arquitetura baseada em componentes (*Component-Based Architecture* – CBA, em inglês) permite a adição de novos componentes com facilidade sem a necessidade de recompilação do código, o que torna a solução bastante flexível. Além disso, o acoplamento entre os componentes é mínimo uma vez que estes obedecem a uma interface bem definida, facilitando, portanto, a manutenção.
	Desvantagens	Projetar uma arquitetura baseada em componentes requer que todo o sistema seja dividido em módulos com interfaces bem definidas. Identificar a decomposição apropriada do sistema não é uma tarefa fácil o que torna a solução potencialmente mais complexa do que soluções monolíticas.
	Observações	Os componentes são interligados usando o padrão *Publish-Subscribe* (*Observer*).

2	Decisão	**Organizar a arquitetura usando o padrão *Model-View-Controller* (MVC)**
	Drivers	Desacoplamento RFs: **[Criação do ambiente de diagnóstico], [Exposição de funcionalidade]** RNFs: **[Testes automáticos]**
	Descrição	A arquitetura deve ser organizada em três grandes módulos: Modelo (M), Visão (V) e Controlador (C). O Modelo representa os dados da aplicação e a lógica de negócio para manipular esses dados. A Visão é a camada de interface com o usuário, onde o usuário vê o estado do modelo e pode manipular a interface, para ativar a lógica de negócio. O Controlador processa e responde a eventos gerados pela visão e pode invocar alterações no modelo, servindo portando de comunicação entre a Visão e o Modelo.
	Vantagens	Permite melhor organizar a interface gráfica e desacoplá-la da lógica de negócio, dando flexibilidade para a existência de várias interfaces com o usuário usando a mesma lógica. Permite também que sejam feitas alterações na interface gráfica sem afetar a lógica de negócio ou vice-versa. Além disso, facilita a exposição da lógica de negócio para possível uso em outras aplicações.
	Desvantagens	
	Observações	É imprescindível que a separação entre interface e lógica de negócio seja rigorosamente observada.
3	Decisão	**Os modelos que descrevem o sistema são descritos na linguagem *Modelica*.**
	Drivers	RFs: **[Criação de modelos], [Interligar modelos de tipos diferentes], [Verificação de consistência de modelos]** RNF: **[Integração com ferramentas de modelagem]**

	Descrição	Os modelos que descrevem os sistemas (que podem ser modelo de sistema ou modelo de sinal) e que são utilizados no processo de diagnóstico de falhas serão descritos através da linguagem de modelagem *Modelica* [Mod]. Chamamos esse modelo de *Modelo do Sistema*.
	Vantagens	• Possibilita a independência de ferramenta de modelagem, uma vez que os modelos podem ser criados/editados em qualquer ferramenta que reconheça a linguagem *Modelica*. • *Modelica* é uma linguagem de modelagem bastante usada tanto pela indústria como por instituições de pesquisas, oferecendo, portanto, um excelente know-how.
	Desvantagens	
	Observações	A *Biblioteca Padrão Modelica* [ModelicaLibrary] e a biblioteca *ExtendedPetriNet* [ExtendedPetriNet] serão disponibilizadas junto com a ferramenta. Com essas bibliotecas é possível criar os vários tipos de modelos elencados no requisito "Criação de modelos". (veja requisito RF 1 no capítulo 4). Também poderão ser disponibilizadas outras bibliotecas que facilitem a construção de modelos.
4	Decisão	**O processo de detecção de falhas será descrito na linguagem *Modelica***
	Drivers	RFs: **[Modelo de sistema e modelo de sinal], [Realização do diagnóstico], [Simulação do diagnóstico]** RNFs: **[Integração com ferramentas de modelagem]**
	Descrição	O processo de detecção de falhas, que inclui a geração e avaliação de resíduos, será descrito em *Modelica*. Chamamos esse modelo de *Modelo de Detecção*.

	Vantagens	• Facilita o desenvolvimento de componentes de diagnóstico além de permitir usar todos os componentes e funções *Modelica* já disponíveis. • Aumenta a flexibilidade na construção do processo de detecção de falhas, possibilitando também uma melhor interação com o modelo do sistema. • Possibilita usar todas as funcionalidades de criação/edição e simulação da ferramenta de modelagem durante a implementação do processo de detecção de falhas.
	Desvantagens	Requer que o usuário que irá implementar o processo de detecção de falhas conheça a linguagem *Modelica*.
	Observações	Um modelo de detecção *default*, pode ser gerado automáticamente.
5	Decisão	**Usar a ferramenta *SimForge***
	Drivers	Princípio "buy, don't build". RFs: **[Criação de Modelos], [Interligar modelos de tipos diferentes], [Adição de novos modelos], [Verificação de consistência de modelos], [Execução de modelos]** RNFs: **[Integração com ferramentas de modelagem]**
	Descrição	Usar a ferramenta *SimForge* [SimForge] para a criação, edição e execução dos modelos a serem usados no processo de diagnóstico.
	Vantagens	O uso de uma ferramenta já pronta (princípio *buy, don't build*) para a criação/edição de modelos irá reduzir significativamente o esforço necessário para desenvolvimento da ferramenta. Com isso, os desenvolvedores podem se concentrar em adicionar valor ao invés de "reinventar a roda".

	Desvantagens	O Auto-FDI perde a característica de uma IDE (*Integrated Development Environment*). A idéia de uma IDE seria permitir implementar em um ambiente único, todo o processo necessário para o diagnóstico, incluindo a criação dos modelos. Outra desvantagem é que se torna necessário uma comunicação entre o Auto-FDI e o *SimForge*.
	Observações	O *SimForge* é uma ferramenta grátis e de código aberto. Ela foi escolhida por ser, dentre as possibilidades de ferramentas grátis, a que melhor atende aos requisitos do projeto.
6	Decisão	**Usar *OpenModelica* para executar e obter meta dados dos modelos**
	Drivers	RFs: **[Realização do diagnóstico], [Simulação do diagnóstico]**
	Descrição	Durante a realização do diagnóstico, os modelos criados no *SimForge* serão executados no software *OpenModelica* [OpenMod], possibilitando que o Auto-FDI seja independente de ferramenta comercial. Meta dados dos modelos (como, por exemplo, variáveis dos modelos, valores dos parâmetros, etc.) também serão obtidos através do *OpenModelica*.
	Vantagens	• *OpenModelica* é um software grátis e de código aberto. A execução do modelo no *OpenModelica* irá permitir que o sistema Auto-FDI seja independente de ferramenta comercial. • *OpenModelica* possui uma API para obtenção de metada dados de modelos modelica, o que irá facilitar o desenvolvimento.
	Desvantagens	• A execução no *OpenModelica* é menos eficiente se comparado com soluções comerciais, como por exemplo, o *Dymola* [Dym]. • A versal atual do *OpenModelica* não dá suporte à versão mais nova da linguagem *Modelica*.
	Observações	

7	Decisão	**Desenvolver a ferramenta usando a plataforma .NET**
	Drivers	RF: **[Criação do ambiente de diagnóstico]** RNF: **[Software básico]**
	Descrição	**Com a finalidade de facilitar a construção de interfaces gráficas, a ferramenta deve ser desenvolvida com a plataforma .Net utilizando a linguagem C#.**
	Vantagens	• A plataforma .Net dispõe de um excelente ferramental de desenvolvimento de software. • Permite criar, mais facilmente, interfaces gráficas sofisticadas e com ótimo apelo visual. • A plataforma dá suporte a uma grande variedade de linguagem, permitindo a interoperabilidade entre as diversas linguagens.
	Desvantagens	• O software poderá ser usado somente no sistema operacional Windows. • .Net dispõe de uma menor quantidade de soluções open source, se comparado com Java.
	Observações	Existem projetos que permitem a portabilidade de aplicações .Net (como, por exemplo, o Mono [Mono]). Porém, não faz parte dos objetivos deste projeto testar a ferramenta em outros sistemas operacionais que não seja o Windows XP.
8	Decisão	**Usar uma *Façade* para acessar a lógica de negócio**
	Drivers	Desacoplamento RNFs: **[Testes automáticos], [Exposição de funcionalidades]**
	Descrição	A camada de lógica de negócio (Modelo) deve ser acessada através de uma Façade, um objeto único. Os métodos da Façade devem permitir acessar toda a lógica de negócio.

	Vantagens	A Façade provê uma interface unificada, de mais alto nível, para um conjunto de interfaces em um subsistema, deixando-o mais fácil de entender e usar. A Façade também pode reduzir dependências em relação às características internas de uma biblioteca, reduzindo o acoplamento e aumentando a ocultação de informação.
	Desvantagens	A classe Façade pode ficar muito grande quando a lógica de negócio é extensa.
	Observações	• A lógica de negócio é estritamente acessada pela Façade. Os clientes que usam a Façade (como por exemplo, o controlador da arquitetura MVC) não têm acesso diretamente à lógica de negócio. • É importante ressaltar também que todo o trabalho é feito pela lógica de negócio e a Façade realiza apenas chamadas aos objetos de negócio apropriados, fazendo um mapeamento de API, se necessário.
9	Decisão	**Usar adaptadores para injetar sinais do sistema no modelo.**
	Drivers	RFs: **[Criação de Modelos], [Execução de modelos], [Realização do diagnóstico]**
	Descrição	Módulos independentes chamados "adaptadores" são utilizados para injetar sinais do sistema nos modelos. Adaptadores são componentes que sabem como acessar esses dados. Estes componentes devem ser o mais genérico quanto possível e podem ser configurados para lidar com uma fonte particular através da escolha adequada de seus parâmetros.
	Vantagens	• O uso de adaptadores ajuda a lidar com a heterogeneidade de fontes com maior facilidade. • O uso de adaptadores genéricos parametrizados permite diminuir o número de (tipos de) adaptadores.
	Desvantagens	Adaptadores genéricos parametrizados são de elaboração mais difícil do que adaptadores particulares.
	Observações	

10	Decisão	**Os adaptadores de fontes de entrada devem ser componentes *Modelica***
	Drivers	RFs: **[Criação de Modelos], [Execução de modelos], [Realização do diagnóstico]**
	Descrição	Os adaptadores de fontes de entrada devem ser componentes descritos na linguagem *Modelica*. A função que faz a leitura dos arquivos (ou de outra fonte) deve ser escrita em *Modelica* ou em linguagens que possam ser chamadas diretamente pelo código *Modelica* (linguagem C ou Fortran).
	Vantagens	• Permite executar os modelos com fontes de dados externas, diretamente na ferramenta de modelagem, sem a necessidade de ter um programa servidor para fornecer esses dados. • Permite configurar os parâmetros desses componentes na ferramenta de modelagem *Modelica*. • Possibilita interligar, estaticamente e visualmente, a saída dos adaptadores aos componentes do modelo *Modelica*. Caso os adaptadores não fossem componentes *Modelica*, isso não seria possível.
	Desvantagens	A equipe de desenvolvimento poderá ter mais dificuldade em criar componentes na linguagem *Modelica* do que se fosse feito em C#.
	Observações	Caso os adaptadores de fontes não fossem componentes *Modelica*, para executar os modelos na ferramenta de modelagem *Modelica* (no *SimForge*, por exemplo), seria necessário ter um programa servidor rodando para fornecer os dados para o modelo. Seria inevitável também que o modelo soubesse da existência desse servidor e fizesse as requisições quando necessitasse dos dados.
11	Decisão	**Componentes especiais para conectar modelos de tipos diferentes.**
	Drivers	RFs: **[Criação de Modelos]**

	Descrição	Componentes *Modelica* especiais chamados “conectores” são usados para interligar modelos de tipos diferentes. Esses componentes recebem um determinado tipo de sinal como entrada (como por exemplo, um valor real) e produz como saída, um sinal de outro tipo (um sinal booleano, por exemplo). Os conectores podem ser configurados a partir de seus parâmetros de modo a ser o mais genéricos possível. Novos conectores poderão ser criados e adicionados na biblioteca de componentes da ferramenta.
	Vantagens	Facilita interligar modelos de tipos diferentes, retirando do modelo a responsabilidade de converter os sinais de entrada/saída.
	Desvantagens	
	Observações	
12	Decisão	**Modelo para injeção de falhas**
	Drivers	RF: **[Simulação do diagnóstico]**, **[Execução de modelos]**
	Descrição	Para fins de validação do diagnóstico de falhas, é possível incluir no *Modelo de Detecção*, em substituição aos adaptadores de fonte de entrada do sistema, um modelo do sistema com componentes de injeção de falhas. Vários componentes de injeção de falhas devem ser disponibilizados, tanto para injeção de falhas aditivas quanto falhas multiplicativas.
	Vantagens	• Permite testar o processo de diagnóstico de falhas sem a necessidade de ter dados reais do sistema. • Facilita a injeção de falhas para verificação do processo de diagnóstico.
	Desvantagens	
	Observações	Os parâmetros dos componentes de injeção de falhas são alterados de mesma maneira que é feito para outro componente do modelo *Modelica*.

13	Decisão	**O tempo de execução é controlado por um relógio universal**
	Drivers	RFs**: [Realização do diagnóstico], [Simulação do diagnóstico]**
	Descrição	Deve ser usado um relógio universal que controla todo o tempo de execução do diagnóstico. Esse relógio gera eventos do tipo *ClockEvent*, indicando a passagem de uma unidade de tempo. Esse evento pode ser observado por qualquer componente que tenha interesse em acompanhar a passagem do tempo (*ClockListener*).
	Vantagens	• Simplicidade e flexibilidade no suporte ao modelo pull. • Possibilita um total sincronismo no tempo de toda a aplicação.
	Desvantagens	Uma desvantagem pode ser a uma redução do desempenho uma vez que pode ser gerada uma grande quantidade de eventos para vários ouvintes. Entretanto essa perda de desempenho não é perceptível ao usuário final da aplicação.
	Observações	Os componentes usam o padrão *Publish-Subscribe* (*Observer*) para observar os eventos do tipo *ClockEvent*.
14	Decisão	**O fluxo de informação segue o modelo Pull.**
	Drivers	RFs: **[Realização do diagnóstico], [Simulação do diagnóstico]**
	Descrição	O fluxo de informação segue o modelo *Pull*, ou seja, quando uma componente necessita de uma informação, este "puxa" a informação de outro componente, que por sua vez também irá "puxar" a informação desejada e assim sucessivamente, formando uma rede de fluxo de informação. O modelo *Pull*, se opõe ao modelo *Push*, onde a informação é "empurrada" para os interessados.

	Vantagens	• O modelo *Pull* tem menor acoplamento do que o modelo *Push*, pois o modelo *Push* supõe que o "source" sabe das necessidades de informação dos "listeners". • Facilita a coleta de informação baseada no tempo, pois com o modelo *Push* seria necessário, por exemplo, verificar se os dados recebidos de diferentes sources estão sincronizados no tempo.
	Desvantagens	
	Observações	
15	Decisão	**Construir um editor para a criação de diagnosticadores.**
	Drivers	RF: **[Editor de interface de diagnosticadores]**
	Descrição	Não usaremos a ferramenta de modelagem para realização do diagnóstico. Ao invés disso, criaremos um editor para a montagem do ambiente onde será realizado o diagnóstico.
	Vantagens	• Podemos criar uma interface gráfica mais sofisticada, personalizada e mais fácil de usar. • O diagnóstico poderá ser feito sem a necessidade de ter a ferramenta de modelagem instalada.
	Desvantagens	Teremos que implementar algumas funcionalidades já providas pelas ferramentas de modelagem, como por exemplo a geração de gráficos.
	Observações	Teremos nosso próprio editor para criação do ambiente de diagnóstico porque com o *SimForge* não é possível obter uma interface altamente personalizada. Também não seria possível obter algumas das características que desejamos (como por exemplo, exibir um alarme na presença de falhas).
16	Decisão	**Usar uma biblioteca para construção do ambiente de diagnóstico**
	Drivers	Princípio "buy, don't build". RF: **[Criação do ambiente de diagnóstico]**

	Descrição	Deve ser usada uma biblioteca pronta (grátis ou não) que permita criar rapidamente aplicações com diagramas (com imagens, figuras, conectores, etc.)
	Vantagens	• Facilidade na criação de diagramas • Utilização de funcionalidades avançadas providas por estas bibliotecas (como por exemplo, zoom in, zoom out, imagens 3D) sem custo adicional de desenvolvimento.
	Desvantagens	Custo adicional, caso a biblioteca escolhida não seja gratuita.
	Observações	Sugerimos que as seguintes ferramentas (entre outras) sejam analisadas: • AddFlow (solução comercial): http://www.lassalle.com/afnet2.htm • CFlowchartEditor (solução grátis de código aberto) : http://www.codeproject.com/KB/miscctrl/flowcharteditor.aspx
17	Decisão	**Uso de *NUnit* para expressar os testes de aceitação**
	Drivers	RNF: **[Testes automáticos]**
	Descrição	Os testes de aceitação serão implementados em C#, usando o framework *NUnit*.
	Vantagens	• Será usado o mesmo *framework* (o *NUnit*) para expressar os testes de aceitação e os testes de unidade. • Permite ter um ambiente de desenvolvimento e de testes totalmente integrados no *VisualStudio*. • Facilidade na localização dos erros e depuração do código com auxílio do *VisualStudio*. • A linguagem C# possui maior poder de expressão para testar do que uma linguagem de script teria.
	Desvantagens	É necessário conhecer a linguagem C# e o framework *NUnit* para implementar o testes de aceitação. Pressupõe-se que o cliente não irá escrever os testes em C#, sendo necessário, portanto, que o programador transcreva os testes expressos pelo cliente para C#.

	Observações	
18	Decisão	**Não haverá testes automatizados para interface gráfica**
	Drivers	RNF: **[Testes automáticos]**
	Descrição	Não haverá testes automáticos para a interface gráfica da aplicação.
	Vantagens	Elimina o trabalho adicional de refazer os testes quando a interface gráfica muda.
	Desvantagens	É necessário fazer uma verificação manual para assegurar que a comunicação com o modelo está sendo feita corretamente. Isto deve ser feito repetitivamente sempre que houver mudança no código.
	Observações	Como o projeto segue o padrão MVC, a camada de interface com o usuário não contém nada da lógica de negócio, tornando-se o mais simples possível. Toda a lógica de negócio está contida em um módulo separado (modelo), que dispõe de testes automáticos (testes de unidade e testes de aceitação).
19	Decisão	**Não será permitida a realização do diagnóstico para vários sistemas simultaneamente**
	Drivers	RFs: **[Realização do diagnóstico], [Simulação do diagnóstico]**
	Descrição	Só será permitido realizar o diagnóstico para um sistema de cada vez.
	Vantagens	Elimina a necessidade de paralelismo, facilitando o desenvolvimento.
	Desvantagens	Só será possível realizar o diagnóstico para um sistema de cada vez.
	Observações	O cliente não expressou interesse em realizar diagnóstico para mais de um sistema simultaneamente. Isso só iria confundir o usuário com uma quantidade excessiva de informação.
20	Decisão	**Uso de Web Services**
	Drivers	RF: **[Exposição de funcionalidades]**
	Descrição	A lógica de negócio será exposta via *Web Services*.

	Vantagens	• As tecnologias subjacentes aos *Web Services* são padrões abertos como HTTP, SOAP, UDDI, ao invés de tecnologias proprietárias; • É mais simples de se implementar que as soluções tradicionais que utilizam CORBA ou DCOM, por exemplo;
	Desvantagens	Uma crítica diz respeito ao fato de o SOAP ser menos eficiente do que os sistemas de chamadas remotas de procimento (RPC) existentes. Por exemplo, as mensagens (com os respectivos envelopes e descrição de tipos) trocadas entre as partes são descritas em formato de texto/XML enquanto que nos sistemas clássicos de RPC são trocadas em formato binário. No entanto, essa desvantagem é compensada significativamente pela facilidade de interoperabilidade entre os serviços, sem os problemas conhecidos de segurança/firewalls, e pela facilidade de se esconder os detalhes proprietários das infra-estruturas de suporte.
	Observações	Referência: en.wikipedia.org/wiki/Web_service

5.3 Estrutura da arquitetura

Nesta seção apresentamos a estrutura estática da arquitetura em termos de seus módulos e interconexões. A apresentação feita em dois passos: primeiramente, na seção 5.3.1, são descritos os módulos principais da arquitetura usando diagramas UML 2.0 e comentários associados a esses diagramas; em seguida, nas seções 5.3.2 e 5.3.3, são apresentadas as interfaces principais dos módulos do sistema.

A estrutura da arquitetura é descrita seguindo uma abordagem "*top-down*", com os diagramas mais gerais apresentados primeiro e detalhamento dos sub-sistemas apresentados em seguida. O comentário que acompanha cada diagrama explica alguns pontos cruciais da organização e oferece explicações sobre as decisões tomadas fazendo referência a decisões arquiteturais da seção 5.2.

5.3.1 Módulos principais

Diagrama: Arquitetura geral

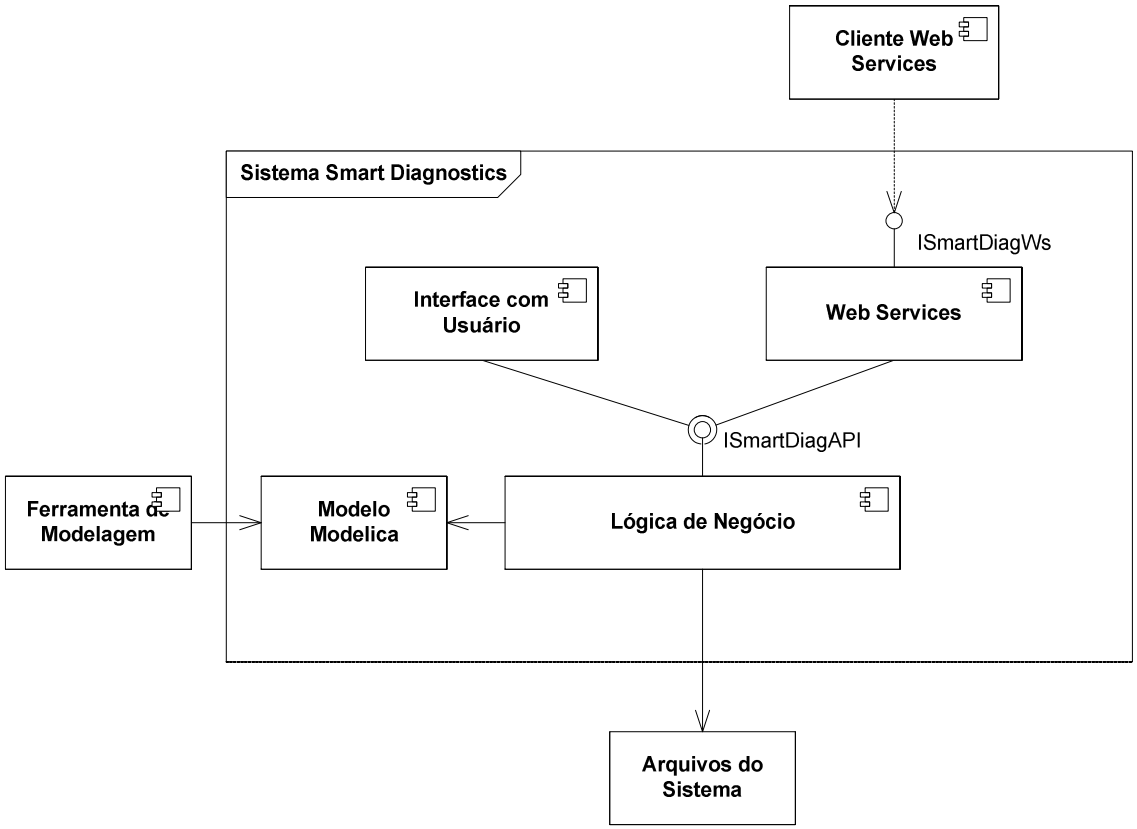


Figura 22: Arquitetura geral do Auto-FDI

Comentário:

Conforme mostra a Figura 22, a arquitetura geral da ferramenta está divida em módulos, cada um isolando uma parte do sistema de outras partes. Os principais módulos são:

- **Interface com o usuário:** Interface desktop que permite acesso aos serviços da lógica de negócio.
- **Lógica de negócio:** Consiste na parte do *Auto-FDI* que implementa os serviços principais do sistema. A lógica de negócio é acessada através de uma interface principal, *ISmartDiagAPI*. Clientes externos podem ser escritos (em linguagens da plataforma .Net) para acessar esta funcionalidade.
- **Web Services:** fornece acesso à lógica de negócio para sistemas externos de forma independente de plataforma e de linguagem de programação. Os serviços são disponibilizados através da interface ISmartDiagWs.

- ***SimForge*:** ferramenta usada para criação/edição e execução dos modelos em *Modelica*, que são os modelos que descrevem o sistema (*Modelos do Sistema*) e os modelos que descrevem o processo de detecção de falha (*Modelos de Detecção*).

Observe que, a parte de teste não foi incluída para não sobrecarregar a figura desnecessariamente.

Diagrama: Interface com o usuário

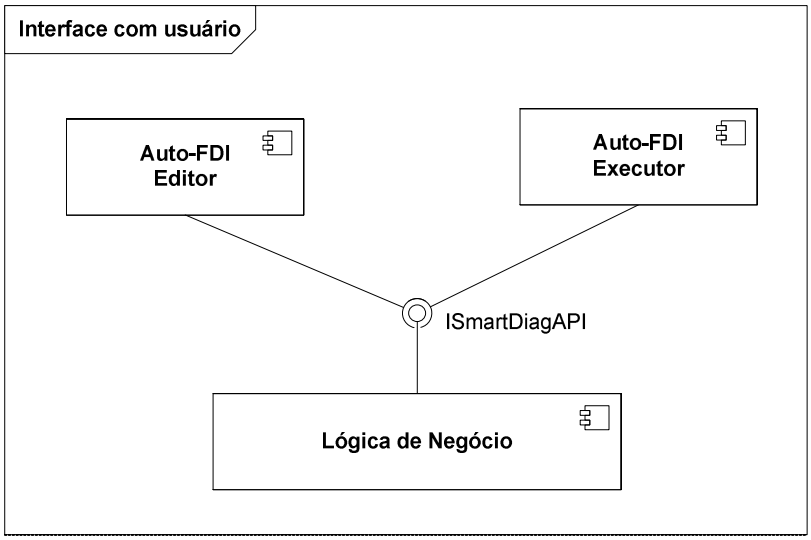


Figura 23: Arquitetura: Interface com o usuário

Comentário:

Conforme já mencionamos na seção 4.2, o Auto-FDI possui três tipos de usuários: o modelador, o criador de diagnosticadores e o operador. Para a criação do modelo foi decidido usar uma ferramenta já existente – o *SimForge* (decisão 5). Para os perfis "criador de diagnosticadores" e "operador" temos duas interfaces gráficas, chamadas respectivamente de Auto-FDI *Editor* e Auto-FDI *Executor*.

A arquitetura é baseada no padrão *Model-View-Controller* (MVC) [GHJV1995], conforme mencionado na *decisão arquitetural* 2. O padrão MVC é ilustrado no diagrama abaixo. O controlador recebe eventos gerados pela interface do usuário (Visão) e mapeia para chamadas à lógica de negócio, exposta através da interface *ISmartDiagAPI*. Ao receber resposta, o controlador atualiza a visão, obtendo os dados da lógica. Alternativamente, mudanças na lógica de negócio podem gerar eventos para a visão que se encarrega de atualizar-se utilizando os dados da lógica de negócio recebidos no evento.

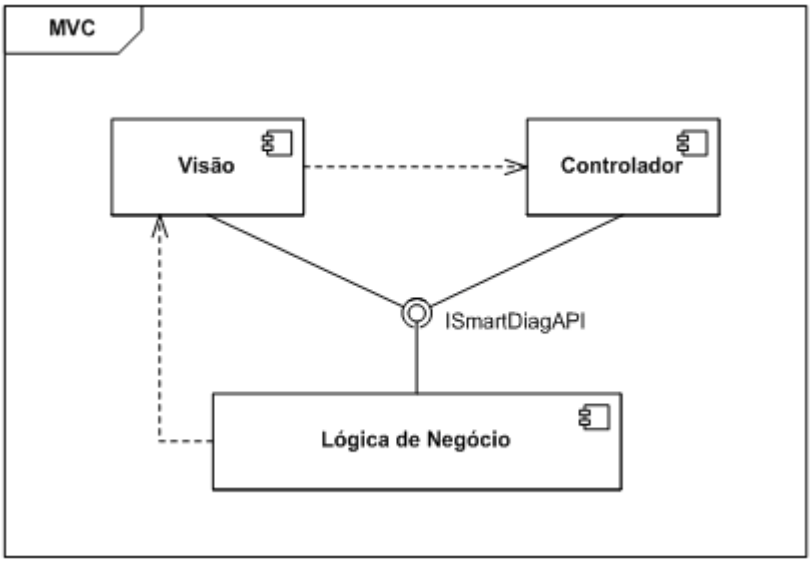


Figura 24: Padrão *Model-View-Controller*

Obs.: as linhas sólidas indicam associação direta e as tracejadas indicam associação indireta (padrão *observer*, por exemplo).

Diagrama: Lógica de negócio

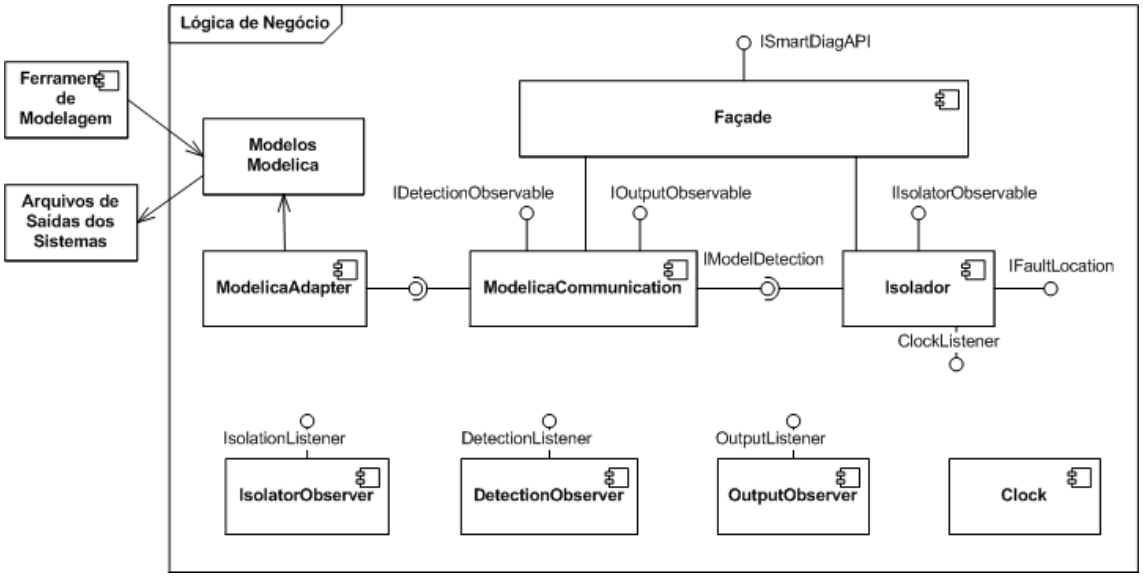


Figura 25: Arquitetura: Lógica de negócio

Comentário:

A lógica de negócio é a parte do Auto-FDI que implementa as principais funcionalidades do sistema. Ela é encarregada de gerar o resultado do diagnóstico através da análise de dados oriundos de um sistema real ou provenientes de simulação. Na lógica de negócio são tratadas questões relacionadas à leitura e ao processamento de

dados dos sistemas, simulação do modelo, controle do tempo da simulação, injeção de falhas, etc. Todas as funcionalidades da lógica de negócio são acessadas estritamente através de uma Façade [GHJV1995], um objeto *Singleton* [GHJV1995]. A Façade provê um ponto de acesso único para a lógica de negócio através da interface *ISmartDiagAPI*. (a decisão de usar uma Façade para acessar a lógica de negócio está explicitada na seção 5.2). Segue uma descrição dos principais módulos da lógica de negócio.

Modelo Modelica: Representa os modelos criados com a ferramenta de modelagem e que serão usados para realização do diagnóstico.

ModelicaProxy: Fornece uma API para fácil acesso à informações sobre o modelo Modelica (variáveis, parâmetros, etc.) e ao resultado da simulação do modelo.

Clock: Este componente gera eventos do tipo *ClockEvent*, indicando a passagem de uma unidade de tempo. Permite cadastrar componentes que tenham interesse em acompanhar a passagem do tempo (*ClockListener*).

ModelicaCommunication: Componente disponibiliza informações sobre o modelo *Modelica* (a estrutura do modelo do sistema, as variáveis, parâmetros, etc.) e sobre resultado da simulação do modelo. Este componente é um *ClockListener*, e portanto, recebe eventos do relógio. Ao receber um evento do relógio, ele coleta informações resultantes da simulação do modelo *Modelica* e produz os eventos *OutputEvent* e *DetectionEvent*. *OutputEvent* permite obter o valor das variáveis de saída do modelo e dos adaptadores de entrada enquanto que *DetectionEvent* permite obter o valor das variáveis dos componentes de detecção. O componente ModelicaCommunication implementa as interfaces *IOutputObservable* e *IDetectionObservable* permitindo que sejam cadastrados interessados em escutar os eventos *OutputEvent* e *DetectionEvent* através do padrão *Observer*.

Isolador: Componente que implementa o algoritmo de isolamento de falhas. É ouvinte do relógio (*ClockListener)*. Ao receber um *ClockEvent*, este componente executa o algoritmo de isolamento de falhas para o tempo em questão e disponibiliza o resultado do isolamento através do evento *IIsolatorEvent*. Este componente implementa a interface *IIsolatorObservable*, permitindo que sejam cadastrados interessados em escutar o eventos do tipo *IIsolatorEvent*.

FacadeSmartDiag: Provê um ponto de acesso único para toda a lógica de negócio através da interface *ISmartDiagAPI*.

Diagrama: Modelo de Detecção

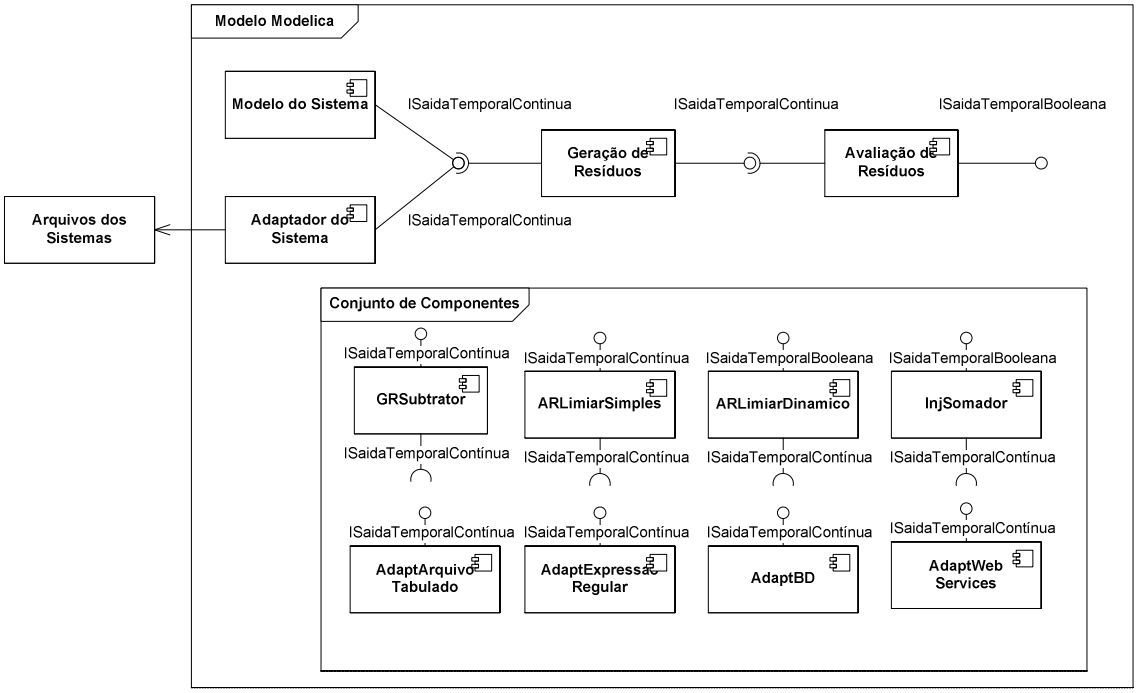


Figura 26: Arquitetura: Modelo de Detecção

Comentário:

Conforme explicitado na seção 5.2 (decisão 4), o Modelo de Detecção (que compreende a geração e avaliação de resíduos) é descrito em *Modelica*. Este modelo é criado por uma ferramenta de modelagem *Modelica*, que no nosso caso, optamos por usar o *SimForge* (decisão 5). Conforme mostra a Figura 26, o Modelo de Detecção contém o Modelo do Sistema e consiste de vários componentes interconectados, cada um dos quais com variáveis de entrada e variáveis de saída. O mecanismo de conexão entre os componentes *Modelica* é explicado na seção 5.3.3.

Vários componentes estão disponíveis para facilitar a criação de Modelos de Detecção. Eles obedecem às interfaces ilustradas na figura acima. Entre eles temos componentes de geração de resíduos, componentes de avaliação de resíduos e os adaptadores, que são componentes usados para acessar os dados do sistema (decisão 9). Os adaptadores e os componentes de geração de resíduos obedecem à interface *ISaidaTemporalContinua* e os componentes de avaliação de resíduo obedecem à interface *ISaidaTemporalBooleana*. Estas interfaces são descritas na seção 5.3.3.

Além desses componentes, existe uma extensa biblioteca de componentes e funções *Modelica* disponibilizadas pela comunidade e que podem ser usados para a construção dos modelos.

Diagrama: Modelo do Sistema

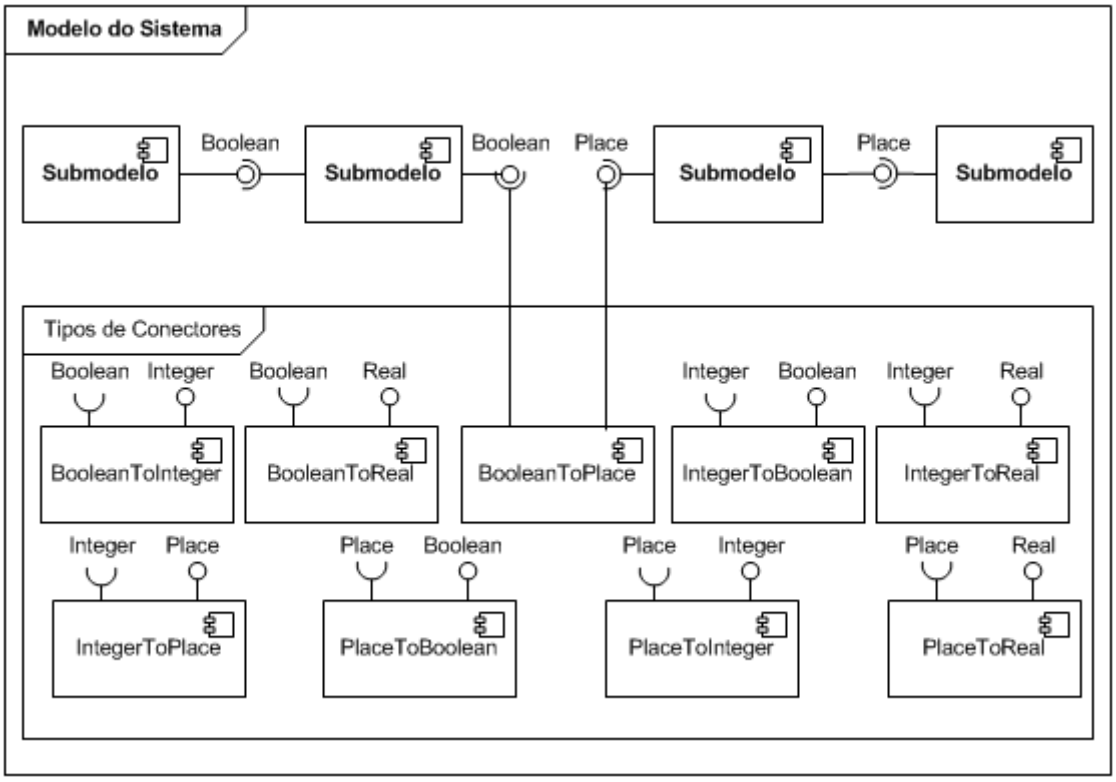


Figura 27: Arquitetura: Modelo do Sistema

Comentário:

O Modelo do Sistema é formado através da composição de submodelos. Submodelos de tipos diferentes são interligados através de um componente especial chamado "conector" (decisão 11). Por exemplo, o conector *BooleanToPlace* permite interligar um modelo do tipo booleano com um modelo representado por redes de Petri. Vários conectores estão disponíveis para interligar diferentes tipos de modelos, como mostra a Figura 27.

Diagrama: Web Services

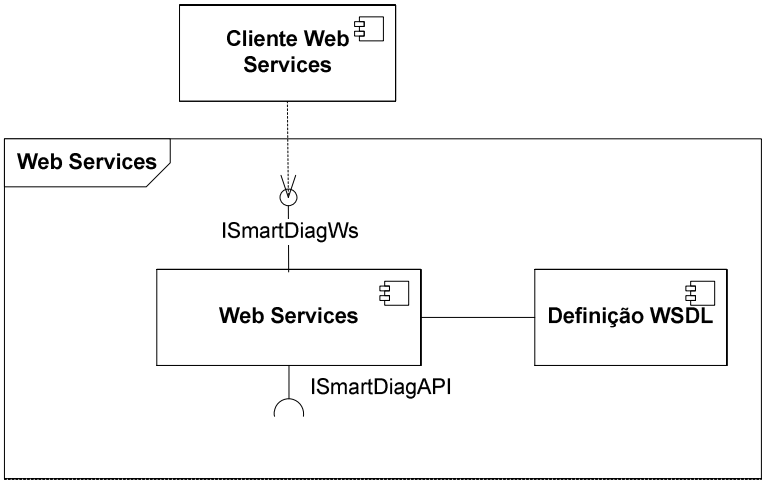


Figura 28: Arquitetura: Web Services

Comentário:

Para facilitar a integração do Auto-FDI com outros sistemas de forma independente de plataforma, a interface *ISmartDiagAPI* é exposta como *Web Services* (decisão 20), gerando a interface *ISmartDiagWs*. A definição do serviço é realizada por um arquivo WSDL (*Web Service Definition Language*). A implementação do *Web Service* pode ser realizada de forma manual ou através de ferramentas que a produzem semi-automaticamente a partir do arquivo WSDL.

5.3.2 Interfaces do sistema

As interfaces importantes do sistema Auto-FDI são descritas a seguir.

Interface	**ClockListener**
Descrição	Permite que um componente receba um evento de relógio.
Serviços	• **clock**(*ClockEvent* event): recebe um evento de relógio.
Implementado por	*Isolador, ModelicaProxy*
Observações	• O que é feito ao receber um evento de relógio depende do componente receptor. Não há semântica associada ao evento de relógio em si; • Um evento de relógio é qualquer evento que contenha um

	timestamp (marcador de tempo).

Interface	**IModelDetection**
Descrição	Permite obter informações sobre o modelo (estrutura do modelo, variáveis, parâmetros, etc.) e sobre resultado da simulação do modelo.
Serviços	• **getModelStructure()** – Retorna um grafo representando a estrutura do modelo (submodelos que compõem o modelo e a conexão entre eles). • **getAllOutputVars()** - Obtém as variáveis de saída de todo o modelo Modelica, incluindo o Modelo do Sistema e o Modelo de Detecção. • **getModelOutputVar**s**()** - Obtém as variáveis de saída apenas do Modelo do Sistema. • **getSystemOutputVars()** – Obtém as variáveis de saída do sistema (dos adaptadores de entrada). • **getSimulationValue(**string var, long time**)** - Obtém o valor de simulação de uma variável do *Modelo Modelica* em um determinado tempo da simulação. • **getDetectionOutputs()** - Obtém as variáveis de saída do Modelo de Detecção. • **getDetectionResult(long time)** - Obtém o resultado do processo de detecção de falha (os sintomas gerados para os componentes do sistema) em um determinado tempo na simulação. • **getAllComponents()** – Obtém os nomes dos componentes do modelo modelica. Inclui componentes do modelo do sistema e da detecção de falhas.

	• **getModelComponents()** – Obtém os nomes dos componentes do modelo Modelica apenas do modelo do sistema. • **getDetectionComponents()** – Obtém os nomes dos componentes do modelo Modelica apenas da parte da detecção de falhas. • **getParameters(**string componente**)** – Obtém os parâmetros de um determinado componente. • **setParameterValue**(string componente, string param, *ObjectValue* value) – Altera um determinado parâmetro de um componente.
Implementado por	*ModelDetection*

Interface	**IFaultLocation**
Descrição	Permite obter informações sobre o resultado do isolamento de falhas.
Serviços	• **getFaults(**long time**)** – Obtém uma coleção com as falhas ocorridas em um determinado tempo da execução do diagnóstico. • **hasFaultOcurred(**string component, long time**)** – Verifica se ocorreu falha em um determinado componente, em um determinado instante de tempo da execução do diagnóstico.
Implementado por	*AutomaticIsolador, BinaryDiagnosticMatrixIsolator*
Observações	A operação **getFaults**(long time) retorna uma coleção de componentes de falhas. Um componente de falha possui alguns atributos, a exemplo do atributo que indica o grau de certeza de ter ocorrido uma falha neste componente.

Interface	**IOutputObservable**
Descrição	Permite adicionar e remover observadores das saídas contínuas dos componentes do modelo Modelica, incluindo as saídas do Modelo do Sistema, dos adaptadores do Modelo de Detecção. Os observadores cadastrados recebem eventos do tipo *OutputEvent*.
Serviços	• **addOutputListener** (*OutputListener* listener) – Adiciona um observador. • **removeOutputListener** (*OutputListener* listener) – Remove um observador cadastrado.
Implementado por	*IModelicaProxy*

Interface	**IDetectionObservable**
Descrição	Permite adicionar e remover observadores do resultado da detecção de falhas, ou seja, os sintomas (valor booleano). Os observadores cadastrados recebem eventos do tipo *DetectionEvent*.
Serviços	• **addDetectionListener**(*DetectionListener* listener) – Adicionar um observador. • **removeDetectionListener**(*DetectionListener* listener) – Remover um observador cadastrado.
Implementado por	*IModelicaProxy*

Interface	**IIsolationObservable**
Descrição	Permite adicionar e remover observadores do resultado do isolamento de falhas. Os observadores cadastrados recebem eventos

	do tipo *IsolationEvent*.
Serviços	• **addIsolationListener** (*IsolationListener* listener) – Adiciona um observador. • **removeIsolationListener** (*IsolationListener* listener) – Remover um observador cadastrado.
Implementado por	Isolador

Interface	**OutputListener**
Descrição	Permite que um componente receba eventos do tipo *OutputEvent*. Ao receber um *OutputEvent* é possível obter o valor das saídas contínuas dos componentes do modelo *Modelica* (incluindo o Modelo do Sistema, adaptadores e o Modelo de Detecção).
Serviços	• **receiveEvent (*OutputEvent* event):** Recebe um evento do tipo *OutputEvent*.
Implementado por	Componente de geração de gráficos
Observações	Utiliza o modelo *Pull* para obtenção da informação.

Interface	**DetectionListener**
Descrição	Permite que um componente receba eventos do tipo *DetectionEvent*. Ao receber um *DetectionEvent* é possível obter o resultado da detecção de falhas, ou seja, os sintomas (valor booleano).
Serviços	• **receiveEvent** (*DetectionEvent* event): recebe um evento do tipo *DetectionEvent*.
Implementado por	Componentes de alarme.

Observações	Utiliza o modelo *Pull* para obtenção da informação.

Interface	**IsolatorListener**
Descrição	Permite que um componente receba eventos do tipo *IsolatorEvent*. Um *IsolatorEvent* permite obter o resultado do isolamento de falhas.
Serviços	• **receiveEvent**(*IsolatorEvent* event): recebe um evento do tipo *IsolatorEvent*.
Implementado por	Componente de exibição de falhas, Componente de Alarmes
Observações	Utiliza o modelo *Pull* para obtenção da informação.

Interface	**SmartDiagAPI**
Descrição	Fornece um ponto de acesso único para toda lógica de negócio do sistema Auto-FDI.
Serviços	• **Load**(xml): Monta o diagnosticador através de uma estrutura especificada em XML. • **Run**(startTime, finishTime) – Realiza o diagnóstico para o intervalo de tempo especificado. • **Stop()** – Finaliza a execução do diagnóstico. • **Pause**() – Pausa a execução do diagnóstico. • **SetSpeed(**speed) – Altera a velocidade da execução. • **GetFaults(**time**) -** Obtém uma coleção com as falhas ocorridas em um determinado tempo da execução.
Implementado por	• Façade

5.3.3 Interfaces dos Componentes Modelica

Os componentes *Modelica* são conectados através do mecanismo de conexão da linguagem *Modelica*. Tais componentes possuem uma interface bem definida, com portas (também chamadas *conectores*) que expõe a interface do componente para mundo externo. Só é permitida a comunicação entre componentes através dos conectores. Conexões entre componentes podem ser estabelecidas através de conectores de tipos equivalentes. A Figura 29 ilustra o mecanismo de conexão da linguagem Modelica.

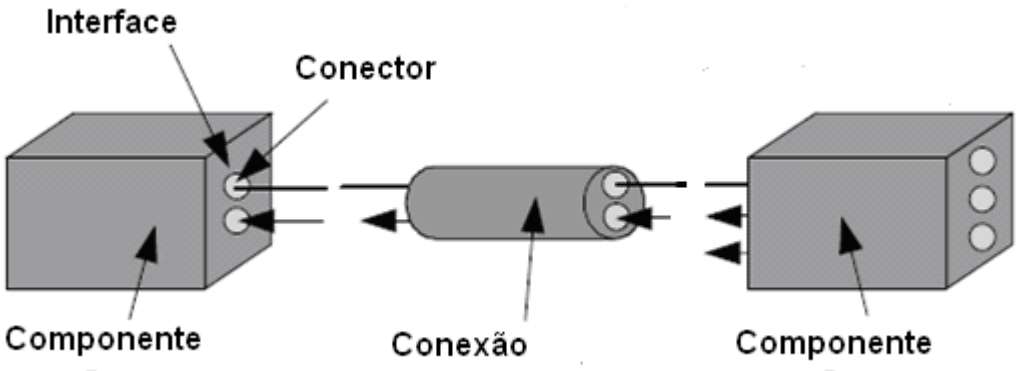


Figura 29: Mecanismo de conexão da linguagem Modelica

Conectores são instâncias de classes do tipo *connector* ou classes do tipo *class* que contempla as restrições de uma classe do tipo connector. Tais conectores declaram variáveis que fazem parte da *interface de comunicação* do componente. Segue abaixo um exemplo de uma classe do tipo conector. Para mais detalhes sobre Modelica e o seu modelo de conexão, consulte a referência [Fritzson2004].

```
Connector Pin
        Voltage v;
        flow Current i;
end Pin;
```

Código 5: Conector da linguagem Modelica

A seguir são descritos os principais componentes *Modelica* e os conectores desses componentes.

Componente	**Componentes de geração de resíduos**

Descrição	Interface para componentes de geração de resíduos.
Conectores	**Variáveis de entrada:** Real modelOutput; Real systemOutput. **Variável de saída**: Real residual.
Implementado por	*Diagnostics.ResidualGenerators.Subtractor*

Componente	**Componentes de avaliação de resíduos**
Descrição	Interface para componentes de avaliação de resíduos.
Conectores	**Variável de entrada:** Real residual. **Variável de saída**: Real symptom.
Implementado por	*Diagnostics.ResidualEvaluators.SimpleThreshold*, *Diagnostics.ResidualEvaluators.DynamicThreshold*

Componente	**Componentes do Modelo do Sistema**
Descrição	Interface para componentes do Modelo do Sistema.
Conectores	**Variável de entrada:** Real residual. **Variável de saída**: Real continuousOutput.
Implementado por	Componentes criados pelo modelador e que são usados na composição do Modelo do Sistema.

Componente	**Componentes Adaptadores de fonte de entrada**
Descrição	Interface para adaptadores de fonte de entrada do sistema.

Conectores	**Parâmetro:** string fileName **Variável de saída:** Real continuousOutput.
Implementado por	*Diagnostics.InputAdapters.TabulatedFileAdapter*, *Diagnostics.InputAdapters.RegularExpressionAdapter,* *Diagnostics.InputAdapters.DataBaseAdapter*,

Componente	**Componentes de injeção de falha**
Descrição	Interface para componentes de injeção de falhas no modelo.
Conectores	**Variável de entrada:** Real continuousTemporalOutput • **Conectores de Saída:** Real continuousTemporalOutput.
Implementado por	*Diagnostics.FaultInjection.AdditiveFault*, *Diagnostics.FaultInjection.MultiplicativeFault*

5.4 Comportamento Dinâmico

Esta seção descreve o comportamento dinâmico do sistema através de alguns cenários do uso do sistema. Os cenários descrevem como a arquitetura responde a determinados estímulos externos. Os cenários são especificados tanto numa visão externa quando numa *visão interna*. Para a *visão externa*, um Caso de Uso de comportamento especifica um conjunto interações entres atores do ambiente e o sistema. Para a visão interna, um modelo de interação mostra como os componentes colaboram para produzir o comportamento desejado.

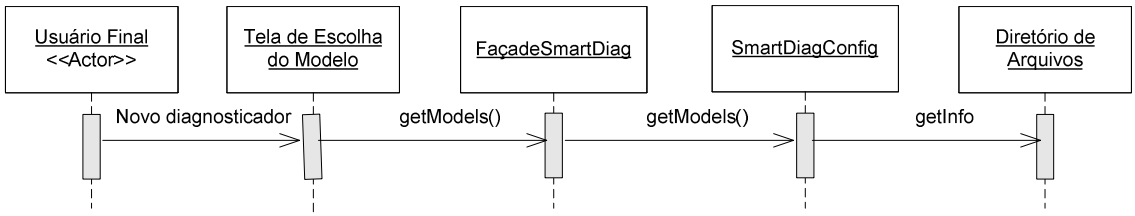
Cenário: Escolha do Modelo do Sistema

Especificação

Caso de Uso	**Escolha do Modelo do Sistema**
Descrição	Ao criar um novo ambiente de diagnóstico, o usuário

	escolhe o modelo do sistema a ser usado durante a realização do diagnóstico.
Atores	Usuário final (criador de diagnosticadores)
Etapas	1. Usuário inicia a criação de um novo ambiente de diagnóstico; 2. É exibida uma tela com a listagem de todos os modelos existentes em um diretório pré-definido da aplicação; 3. O usuário escolhe o modelo a ser usado no diagnosticador.

Modelo de interação

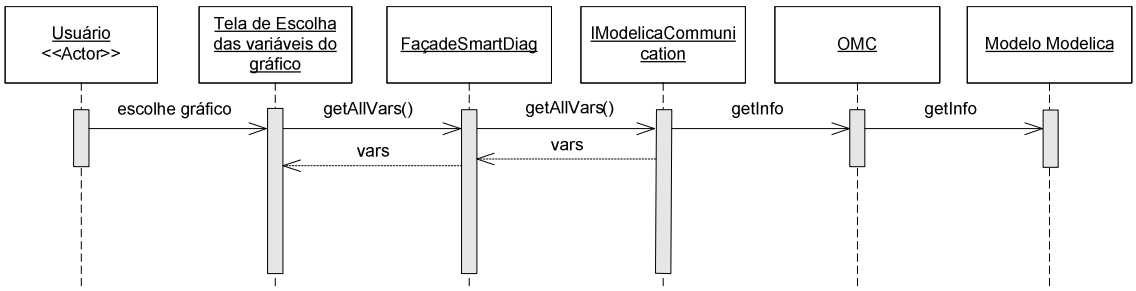


Cenário: Escolha das variáveis a serem exibidas nos gráficos

Especificação

Caso de Uso	**Escolha das variáveis a serem exibidas nos gráficos**
Descrição	Durante a criação/edição de uma ambiente de diagnóstico é possível adicionar gráficos que serão exibidos durante a realização do diagnóstico. Ao adicionar um gráfico, o usuário define as variáveis que serão exibidas no mesmo.
Atores	Usuário Final (criador de diagnosticadores)
Etapas	1. É exibida uma lista de todas as possíveis variáveis (do Modelo do Sistema e do Modelo de Detecção) 2. O usuário escolhe quais as variáveis que serão exibidas no gráfico.

Modelo de interação

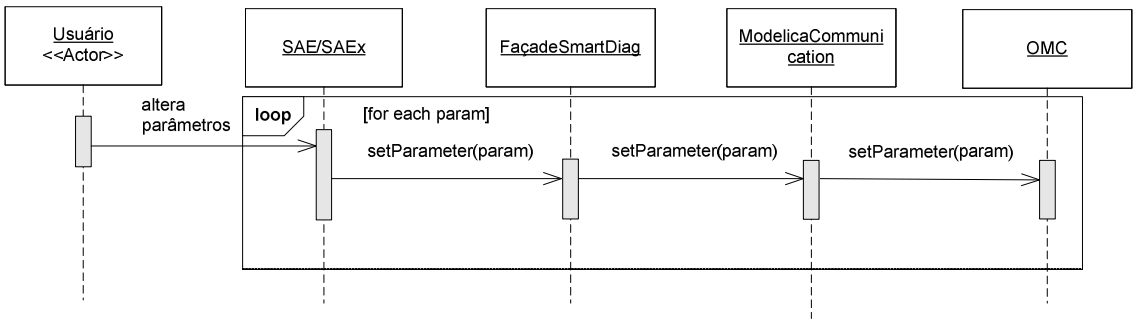


Cenário: Alteração dos parâmetros dos componentes do modelo Modelica

Especificação

Caso de Uso	**Alteração dos parâmetros dos componentes do modelo Modelica**
Descrição	Os parâmetros dos componentes do modelo Modelica são alterados pelo usuário através da interface gráfica (no *Editor* e/ou *Executor*).
Atores	Usuário Final
Etapas	1. Usuário informa os novos valores dos parâmetros na interface gráfica do *Editor* ou do *Executor*; 2. As alterações dos parâmetros são refletidas no modelo.

Modelo de interação

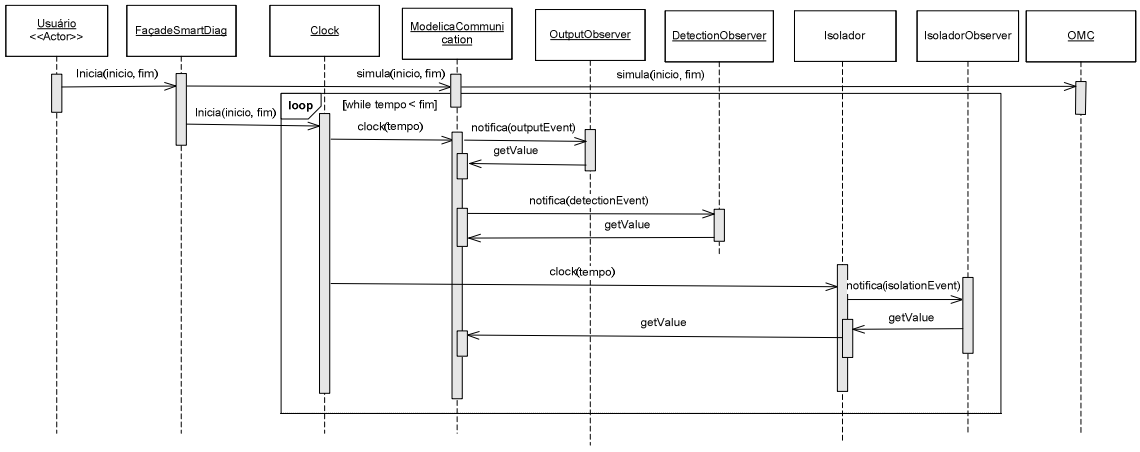


Cenário: Execução do diagnóstico

Especificação

Caso de Uso	**Execução do diagnóstico**
Descrição	O usuário (operador) inicia a realização do diagnóstico. Durante a execução, o resultado do diagnóstico é exibido nos elementos visuais do Executor (gráficos, alarmes, tela de exibição falha, etc.). Para iniciar o diagnóstico o usuário deve informar o intervalo de tempo da simulação (valores de tempo inicial e final).
Atores	Usuário Final
Etapas	1. O usuário informa o intervalo de tempo da simulação e inicia o diagnóstico; 2. O usuário inicia a realização do diagnóstico; 3. O modelo *Modelica* é simulado no *OpenModelica* para o intervalo de tempo fornecido; 4. A cada instante de tempo é lançado um evento do relógio. A cada evento do relógio os valores a serem exibidos são coletados através do modelo *Pull*; 5. Os valores são exibidos nos elementos visuais.

Modelo de interação

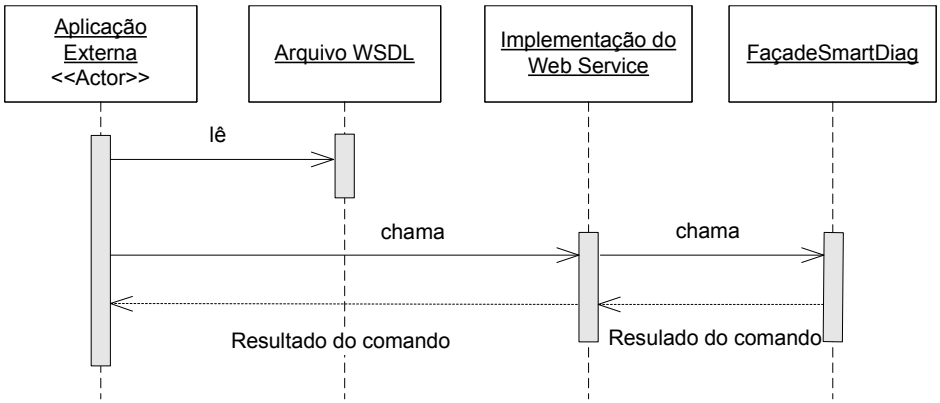


Cenário: Acesso externo ao sistema via *Web Services*

Especificação

Caso de Uso	**Acesso externo ao sistema via *Web Services***
Descrição	A funcionalidade do sistema Auto-FDI é acessada por um sistema externo
Atores	Sistema externo (primário)
Etapas	1. O sistema externo obtém a definição do serviço; 2. O sistema externo chama um serviço do sistema Auto-FDI via protocolo SOAP; 3. Resultados da chamada são recebidos pelo sistema externo

Modelo de interação



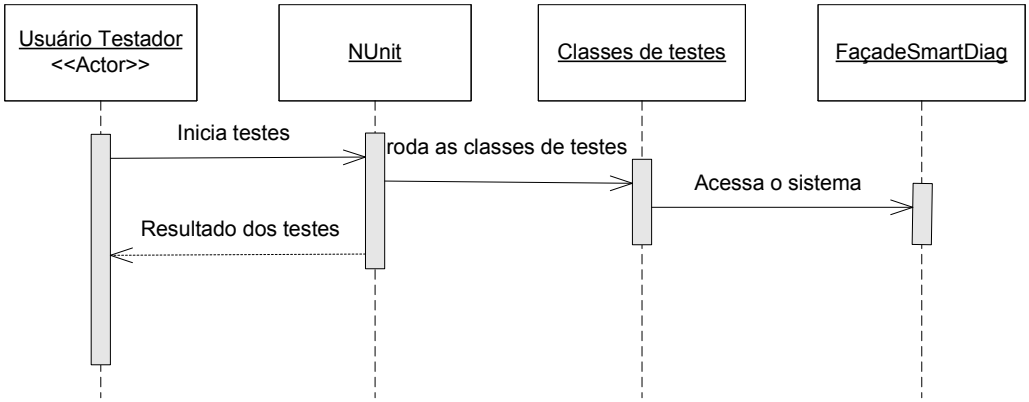
Execução de testes de aceitação

Especificação

Mecanismo	**Execução de testes de aceitação**
Descrição	Testes automáticos são realizados por um testador para verificar se as funcionalidades do sistema foram devidamente implementadas.
Atores envolvidos	Usuário testador (primário)
Etapas	1. O usuário testador inicia os testes da lógica de negócio usando o programa *NUnit*; 2. Os resultados dos testes são exibidos para o

	testador;

Modelo de interação



6 Projeto detalhado e aspectos de implementação

Capítulo 6

Este capítulo tem como objetivo apresentar o projeto detalhado e aspectos de implementação da ferramenta Auto-FDI. No discorrer do capítulo serão apresentadas considerações sobre a interface com o usuário da ferramenta (seção 6.1), sobre as bibliotecas *Modelica* disponibilizadas pela ferramenta para construção de modelos (seção 6.2), e por fim, sobre a organização da ferramenta em estrutura de pacotes de software e as principais classes e interfaces que compõe cada pacote (seção 6.3).

6.1 Interface com o usuário

O projeto cuidadoso de interface com o usuário é uma parte essencial de todo o processo de projeto de software [Sommerville2004]. Uma interface mal projetada pode levar à rejeição de um sistema, comprometendo o sucesso final do projeto. A interface com o usuário do Auto-FDI foi projetada levando em consideração os princípios básicos de projeto, tais como: facilidade de uso, diversidade de usuário, recuperação de erros, documentação interna, entre outros.

Conforme documentado durante o projeto da arquitetura (capítulo 5), a interface com o usuário do Auto-FDI consiste de três ambientes gráficos distintos, uma para cada perfil de usuário. Para a criação dos modelos usados no processo de diagnóstico, optou-se por usar o *SimForge*, uma ferramenta já existente. Para a criação do ambiente de diagnóstico e para a realização do diagnóstico foram desenvolvidas duas interfaces gráficas chamadas respectivamente de Auto-FDI *Editor* e Auto-FDI *Executor*. Ambas foram desenvolvidas usando a tecnologia *Windows Forms*, que é um conjunto de bibliotecas da plataforma .Net. para a criação de interfaces gráficas desktop.

A tela inicial da ferramenta Auto-FDI, ilustrada na Figura 30, permite a escolha entre os diferentes ambientes gráficos. As subseções seguintes descrevem cada um desses ambientes.

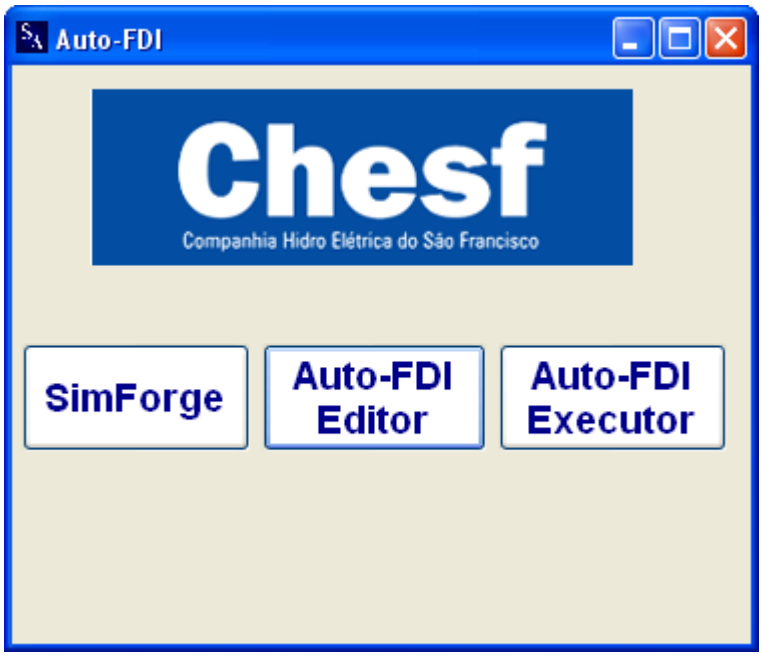


Figura 30: Tela inicial da ferramenta Auto-FDI

6.1.1 *SimForge*

O *SimForge* é uma ferramenta gratuita, de código aberto, que permite a modelagem e simulação de sistemas dos mais diversos campos da engenharia, tais como: mecânica, elétrica, hidráulica, termodinâmica, controle, etc. Os modelos são criados usando a linguagem de modelagem *Modelica*. Internamente, o *SimForge* utiliza o compilador *OpenModelica*, o mesmo compilador *Modelica* usado pelo Auto-FDI.

Com o *SimForge* é possível modelar sistemas de forma rápida e intuitiva através da composição de modelos em um editor gráfico. Modelos e funções estão disponíveis através de uma biblioteca de componentes altamente reusáveis. Ele já vem com a *Biblioteca Padrão Modelica* (*Modelica Standard Library*), uma extensa biblioteca com mais de 780 modelos de componentes genéricos e 550 funções multipropósito, além de permitir facilmente a extensão de sua biblioteca de componentes. Essa biblioteca será descrita na seção 6.2.1. Em adição à *Biblioteca Padrão Modelica*, o Auto-FDI disponibiliza outras bibliotecas para facilitar a criação de modelos utilizados no diagnóstico, a saber: *ExtendedPetriNet*, *ObjectStab* e *Diagnostics,* descritas nas seções 6.2.3, 6.2.2 e 6.2.4, respectivamente.

Embora a criação dos modelos possa ser feita usando qualquer outra ferramenta que dê suporte à linguagem *Modelica*, como por exemplo, *Dymola* [Dym] ou *MathModelica* [MathMod], é importante ressaltar que o Auto-FDI utiliza o

OpenModelica para compilar e executar os modelos internamente. Dessa forma, é possível que modelos criados em ferramentas que utilizem outro compilador não executem no Auto-FDI, devido à incompatibilidade de versão da linguagem *Modelica*.

A Figura 31 ilustra a tela principal do *SimForge*. No lado esquerdo encontra-se a ***janela de navegação de componentes***, que permite acessar os componentes *Modelica* disponíveis para montagem dos modelos. Esses componentes são arrastados para a ***área de montagem do modelo***, na área central da tela.

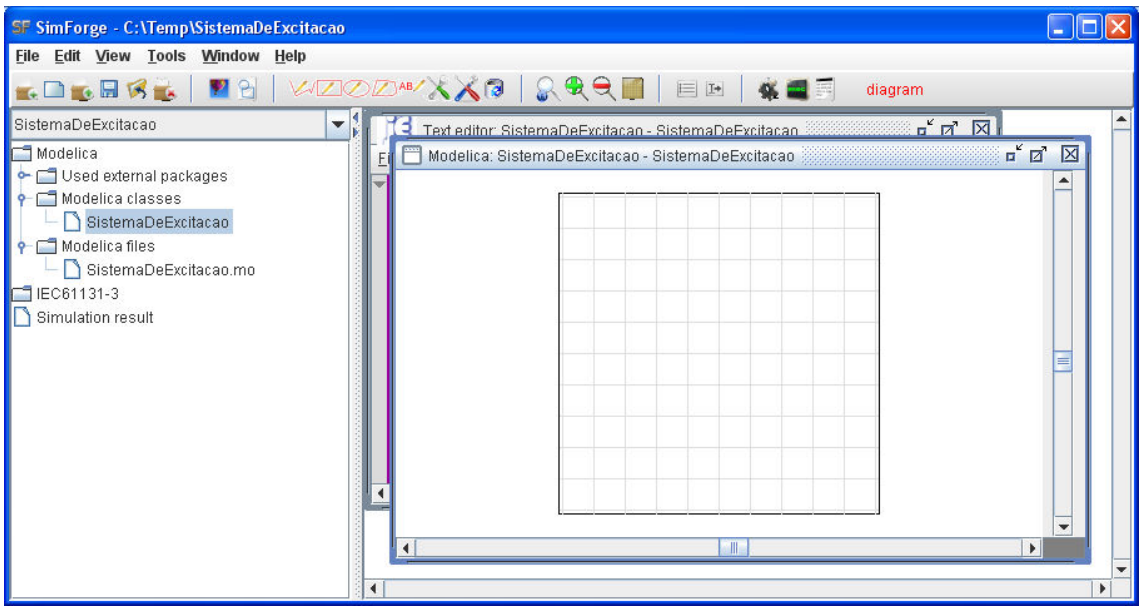


Figura 31: Tela principal do SimForge

Mais detalhes sobre o *SimForge* são fornecidos no apêndice C deste documento.

6.1.2 *Auto-FDI Editor*

O *Auto-FDI Editor,* ou simplesmente *Editor*, é a interface para a criação do ambiente de diagnóstico. O ambiente de diagnóstico é a interface usada pelos operadores para a realização do diagnóstico de sistemas. O *Editor* permite, entre outras funcionalidades:

- Criar uma interface altamente personalizada de acordo com o sistema a ser avaliado. A interface pode incluir imagens do sistema e equipamentos, figuras, links, etc.;
- Adicionar gráficos para observar todo o processo de diagnóstico (valores entradas e saídas do sistema, resíduos, sintomas, falhas, etc.).
- Adicionar alarmes indicadores de falhas ou de sintomas de falha;

- Alteração dos parâmetros dos modelos (do sistema e de detecção de falha) e escolha de quais desses parâmetros poderão ser visíveis e/ou editáveis no ambiente de execução do diagnóstico;

A Figura 32 ilustra a tela principal do *Editor*. No lado esquerdo encontra-se uma ***barra de ferramentas*** (ou toolbar) com componentes gráficos que podem ser arrastados para a ***área de edição***, na parte central. A área de edição pode conter várias abas, permitindo uma melhor disposição dos elementos gráficos. A barra de ferramentas inclui componentes de imagens, *labels*, links, gráficos, e alarmes. No lado direito da tela encontra-se a ***janela de parâmetros***, onde são listados todos os parâmetros do Modelo Modelica. O valor inicial dos parâmetros são aqueles que foram definidos durante a criação desses modelos no *SimForge*. Nesta janela é possível alterar o valor desses parâmetros e definir quais destes parâmetros poderão ser visíveis e/ou editados posteriormente, no *Auto-FDI Executor*.

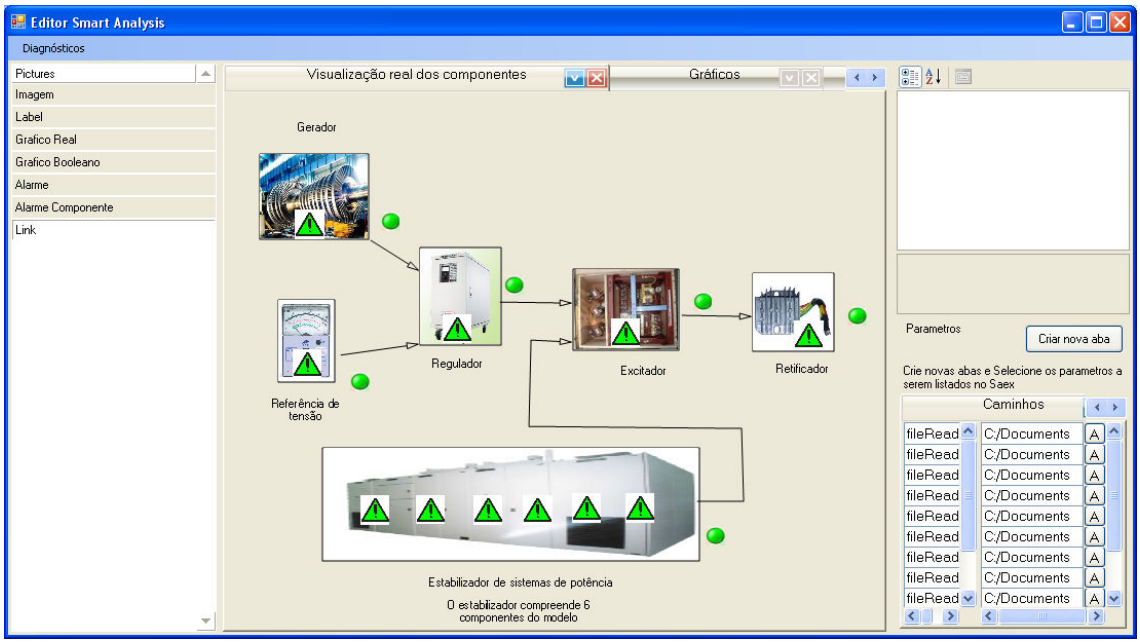


Figura 32: Tela principal do Auto-FDI Editor

Na área de representação do sistema, em destaque na Figura 33 estão os componentes de alarmes.

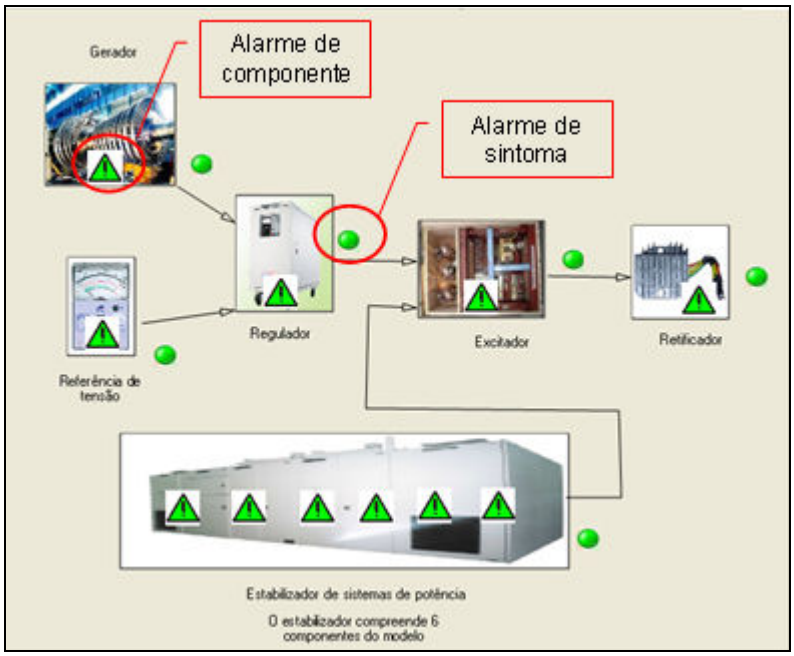


Figura 33: Auto-FDI Editor – Componentes de alarmes

Os ***alarmes de sintoma,*** como o próprio nome sugere, representam os sintomas gerados durante a detecção de falhas. Os sintomas são indícios da presença de falha no sistema. Durante a realização do diagnóstico, um alarme irá assumir a cor vermelha no intervalo de tempo em que componente do sistema ao qual ele está associado apresentar sintomas de falha.

Os ***alarmes de componente*** indicam que um determinado componente do sistema (o componente de sistema a qual esse componente está associado) é uma fonte potencial de falhas. Os alarmes de componentes também assumem a cor vermelha na presença de falhas durante o diagnóstico.

A diferença entre alarmes de sintoma e alarmes de componentes pode ser evidenciada ao observar a Figura 34, obtida durante a execução do diagnóstico no Executor. Como podemos observar na figura, somente o Gerador é realmente uma fonte potencial de falha (estado vermelho). Isso porque os sintomas de falha apresentados pelos outros equipamentos (Regulador, Excitador e Retificador) são possivelmente devido à propagação da falha do Gerador nestes componentes.

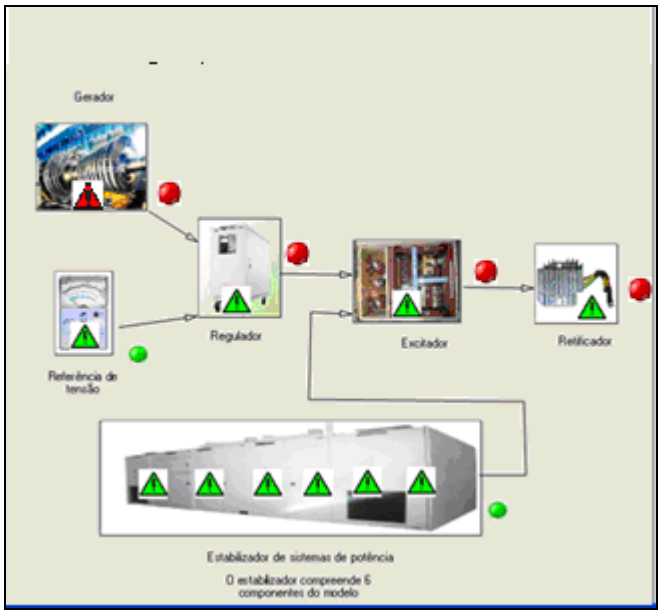


Figura 34: Auto-FDI Editor – Diferença entre os componentes de alarme

Mais detalhes sobre o *Auto-FDI Editor* podem ser obtidos no apêndice C deste documento.

6.1.3 *Auto-FDI Executor*

O Auto-FDI *Executor* (ou simplesmente *Executor)* é a interface criada com o *Editor* e onde de fato é realizado o diagnóstico. A Figura 35 mostra a tela principal do *Executor*, obtida durante a realização do diagnóstico em um determinado sistema. No lado esquerdo da tela são listados os parâmetros do modelo que foram escolhidos durante a montagem da interface no Editor. Os parâmetros que não estiverem na aba "Todos" poderão ser alterados em cada execução do diagnóstico. Na parte superior encontra-se a ***barra ferramenta de execução*** do diagnóstico, que permite iniciar, pausar e parar a execução. Também existem opções para definir o tempo de simulação, o intervalo de amostragem de dados e o controle da velocidade de simulação. No lado direito da tela temos a ***janela de exibição de falhas***. Nessa janela são exibidos detalhes sobre os componentes do sistema que apresentarem falha. Observe também na figura os componentes de alarmes indicando a presença de uma falha no sistema (no Gerador) e a propagação da falha através dos componentes interligados ao Gerador.

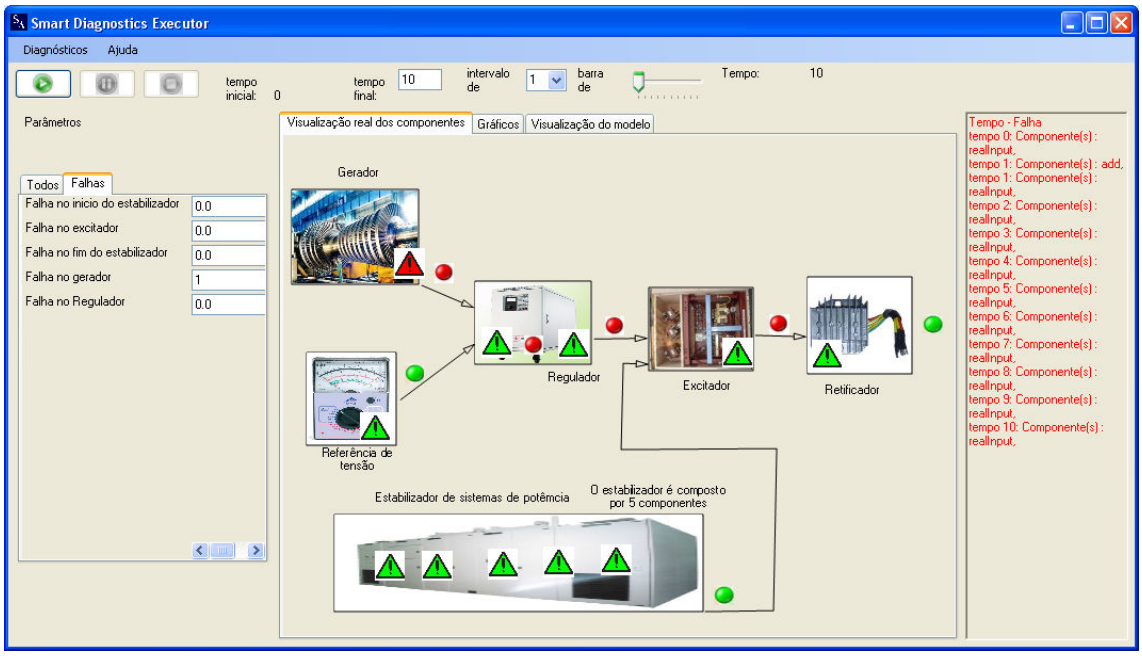


Figura 35: Tela principal do Auto-FDI Executor

Mais detalhes sobre o *Auto-FDI Executor* podem ser obtidos no apêndice C deste documento.

6.2 Bibliotecas de componentes Modelica

Conforme já mencionamos, o Auto-FDI disponibiliza algumas bibliotecas *Modelica* para facilitar a criação dos modelos usados no diagnóstico. São elas: *Biblioteca Padrão Modelica, ObjectStab, ExtendedPetriNet* e *Diagnostics*. Nesta seção descrevemos sucintamente essas bibliotecas. As três primeiras são bibliotecas já existentes, disponibilizadas pela comunidade *Modelica*. A última, a biblioteca *Diagnostics*, reúne um conjunto de componentes que foram criados durante este trabalho. Entre os componentes dessa biblioteca estão componentes de geração de resíduos, de avaliação de resíduos, de injeção de sinais, de injeção de falhas, etc.

O conjunto de biblioteca fornecido pelo Auto-FDI permite a criação dos vários tipos de modelos elencados no requisito "Criação de modelos". (veja requisito RF 1 no capítulo 4), além de facilitar a criação do Modelo de Detecção.

6.2.1 *Biblioteca Padrão Modelica*

A *Biblioteca Padrão Modelica* é uma biblioteca disponibilizada gratuitamente pela organização *The Modelica Association.* A biblioteca contém os componentes mais

comumente usados na modelagem de sistema de vários domínios da engenharia, tais como: mecânica, elétrica, hidráulica, termodinâmica, controle, etc. Atualmente ela dispõe de mais de 780 modelos de componentes genéricos e 550 funções multipropósito (referência à versão 3.0 de fevereiro de 2008), distribuídos nos seguintes pacotes:

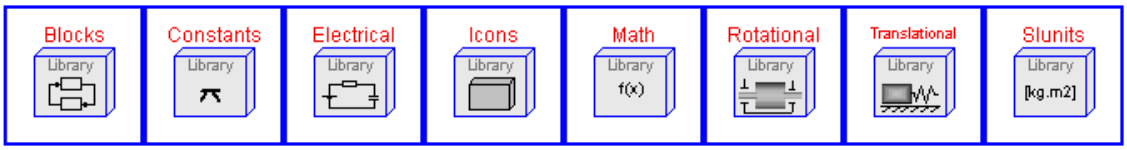


Figura 36: Pacotes da Biblioteca Padrão Modelica

Abaixo segue uma breve descrição de alguns dos pacotes da hierarquia de pacotes da *Biblioteca Padrão Modelica*.

- Modelica.Electrical.Analog – Componentes elétricos e eletrônicos analógicos tais como: resistor, capacitor, transformadores, diodos, transistores, linhas de transmissão, sensores, entre outros;
- Modelica.Electrical.Digital – Componentes elétricos digitais, incluindo *delays*, *gates*, etc.;
- Modelica.Electrical.Machines – Máquinas elétricas, tais como motores e geradores assíncronos, síncronos e de corrente contínua (DC);
- Modelica.Mechanics.Rotational – Sistemas mecânicos rotacionais de 1-dimensão, como por exemplo, caixa-de-marcha, embreagem;
- Modelica.Mechanics.Translational – Sistemas mecânicos transacionais de 1-dimensão, tais como massa, mola e força;
- Modelica.Mechanics.MultiBody – Sistemas mecânicos de 3-dimensões, como por exemplo, juntas, força e sensores;
- Modelica.Media – Componentes para modelagem de substâncias fluidas;
- Modelica.Thermal – Componentes para modelagem de sistemas termo-fluidos, especialmente para sistemas de resfriamento de máquina. Contém canos, bombas, válvulas, etc.;
- Modelica.Blocks – Blocos de entrada e saída contínuas e discretas. Contém funções de transferência, sistemas lineares, não-lineares, matemáticos, lógicos, entre outros;

- Modelica.StateGraph – Máquinas de estado hierárquicas com poder de modelagem semelhante aos *StateCharts* (diagramas de estados com diversas representatividades);
- Modelica.Math.Matrices / Modelica.Utilities – Funções para operações de matrizes, strings, *streams* e arquivos.
- Modelica.Constants, Modelica.Icons, Modelica.SIunits – Pacote utilitário que provê: constantes freqüentemente usadas, tais como *e*, π, R; ícones que podem ser usados nos modelos; cerca de 450 tipos predefinidos, tais como *Mass*, *Angle*, *Time*, etc.;

6.2.2 *ObjectStab*

ObjectStab [ObjectStab] é uma biblioteca *Modelica* gratuita para modelagem de sistemas de potência. Essa biblioteca foi de grande utilidade para o projeto *Smart Analysis*, pois facilitou a modelagem de sistemas da Chesf. A biblioteca contém modelos de geradores, linhas de transmissão, sistemas de reação ativa (tais como capacitores e atuadores), transformadores, sistemas de excitação de geradores, entre outros. A biblioteca consiste dos seguintes pacotes:

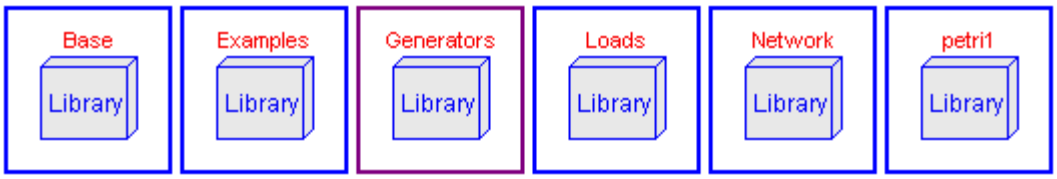


Figura 37: Pacotes da biblioteca *ObjectStab*

Abaixo segue uma breve descrição de cada pacote.

- ObjectStab.Base – Contém a definição comum a todos os subpacotes, incluindo as unidades de grandezas (p.u.);
- ObjectStab.Examples – Provê vários exemplos de modelos de sistemas de potência;
- ObjectStab.Generators – Contém o modelo de vários tipos de geradores;
- ObjectStab.Loads – Contém a definição de tipos de modelos de carga.

- ObjectStab.Network – Contém o modelo de vários componentes de rede, tais como, linha de transmissão, disjuntor, etc.;
- ObjectStab.petri1 – Pacote auxiliar para a criação de modelos usando o formalismo de redes de Petri.

6.2.3 *ExtendedPetriNet*

ExtendedPetriNet [ExtendedPetriNet] é uma biblioteca *Modelica* gratuita especializada em modelagem de sistemas usando o formalismo de redes de Petri. Ela foi adicionada ao Auto-FDI devido ao requisito "[Criação de Modelos]", que especifica que deve ser possível criar modelos representados por redes de Petri. Redes de Petri é uma poderosa técnica de modelagem e especificação de sistemas e que possui uma grande área de aplicabilidade, especialmente na modelagem de sistemas em que o sincronismo entre os eventos é essencial [CV1997].

A *Biblioteca Padrão Modelica* já dispõe de um pacote para a modelagem de sistemas usando redes de Petri que implementa o formalismo de redes de Petri com prioridades determinísticas (pacote *Modelica.StateGraph*). Este tipo de redes de Petri é adequado, por exemplo, para especificação de um sistema de controle, mas tem limitado poder de expressão em outros domínios de problemas [CV1997].

A biblioteca *ExtendedPetriNet* estende a biblioteca de redes de Petri da *Biblioteca Padrão Modelica*, com:

- Transições que permitem atrasos estocásticos ou determinísticos antes de disparar;
- Lugares (*places*) capazes conter mais de um *token*.

Com isso, redes de Petri estocásticas podem ser modeladas. A Figura 38 ilustra os pacotes da biblioteca *ExtendedPetriNet*.

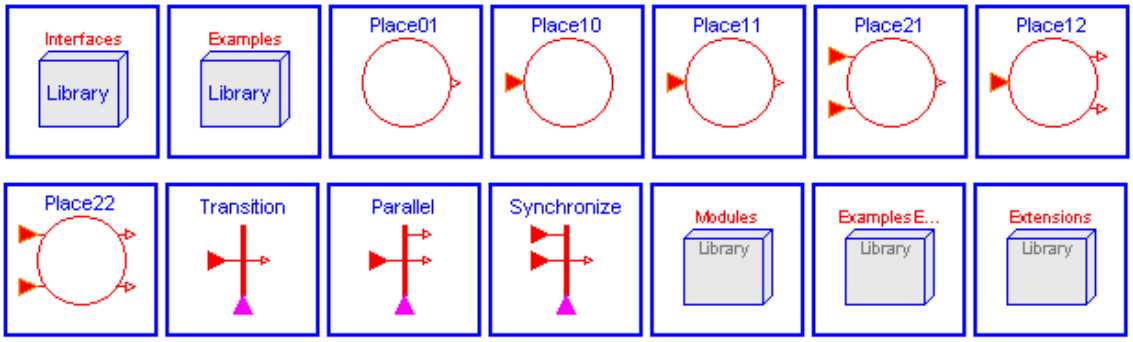


Figura 38: Pacotes da biblioteca ExtendedPetriNet

6.2.4 *Diagnostics*

A biblioteca *Diagnostics* reúne um conjunto de componentes que foram criados durante este trabalho com o propósito de facilitar a construção de modelos usados para realização do diagnóstico. Entre os componentes desta biblioteca estão componentes de geração de resíduos, de avaliação de resíduos, de injeção de sinais, de injeção de falhas, entre outros. Outros componentes poderão ser adicionados a esta biblioteca ao longo da evolução do software. A Figura 39 ilustra os pacotes da biblioteca *Diagnostics*.

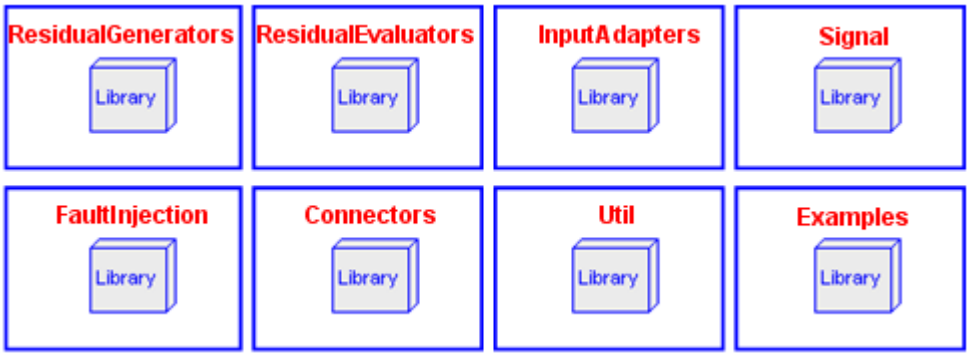


Figura 39: Pacotes da biblioteca Diagnostics

Abaixo segue uma descrição dos pacotes da biblioteca *Diagnostics*.

- *ResidualGenerators* – Contém componentes de geração de resíduos para detecção de falhas, como por exemplo, o componente *Subtractor*, que gera resíduos através de uma simples subtração entre valores obtidos do sistema e valores obtidos do modelo do sistema. Outros componentes de grande utilidade e que poderão ser adicionados futuramente neste pacote inclui: componentes de geração de resíduos usando observadores de estado, filtro de

Kalman, estimativa de parâmetros, entre outros. A fase de detecção de falhas é descrita na seção 2.3.1.

- *ResidualEvaluators* – Contém componentes de avaliação de resíduos para detecção de falhas. Componentes de avaliação de resíduos recebem como entrada os resíduos e produz como saída os sintomas. Entre os componentes deste pacote temos o *SimpleResEvaluator* e o *RobustResEvaluator*. No primeiro, os sintomas são gerados a partir de uma simples comparação do valor absoluto do resíduo com um limiar; no segundo, os sintomas são gerados não com base no valor instantâneo dos resíduos, mas baseado no valor médio dos N últimos valores de resíduos. Exemplos de outros componentes que possivelmente serão adicionados neste pacote incluem: componentes de avaliação de resíduos usando redes neurais e lógica fuzzy. A fase de detecção de falhas é descrita na seção 2.3.1.

- *InputAdapters* – Contém componentes (chamados adaptadores) para leitura dos sinais de entrada e saída do sistema de modo a permitir a injeção desses sinais no modelo. Os adaptadores são componentes que sabem como acessar esses dados. Um exemplo de um componente adaptador de entrada usado constantemente nos modelos construídos neste trabalho é o *TabulatedFileReader*. Este componente lê arquivos em um formato específico, onde cada linha do arquivo é dividida em duas colunas separadas por caractere de espaço. A primeira coluna contém valores numéricos, representando os tempos de medição. A segunda contém os valores das variáveis medidos ao longo tempo. Exemplos de outros componentes que possivelmente serão adicionados neste pacote incluem: adaptadores que obtém dados de arquivos de variados formatos (COMTRADE, por exemplo, muito utilizado em sistemas de potência), de banco de dados, de WebServices, etc.

- *Signal* – Inclui modelos de sinais que poderão ser usados para implementar o processo de detecção de falhas. Métodos de detecção de falhas baseados em modelos de sinais levam em consideração características dos sinais de saída do sistema (como por exemplo, freqüência, amplitude). Entre os modelos de sinais incluem: filtros passa-faixa (*Bandpass filters*, em inglês), análise de

espectro (*Spectral analysis*, em inglês) e estimativa de entropia máxima (*Maximum-entropy estimation*, em inglês).

- *FaultInjection* – Contém componentes especiais que permite a injeção de falhas no modelo, usados para fins de verificação e validação do processo de diagnóstico. Componentes de injeção de falhas permitem alterar os valores de entradas e/ou saída de componentes do modelo *Modelica*, simulando dessa forma, falhas nos componentes do sistema. Os componentes de injeção de falhas disponibilizados neste pacote permitem tanto a injeção de falhas aditivas quanto falhas multiplicativas. Exemplos de componentes desse pacote são: *AdditiveFault* e *RandomAdditiveFault* para injeção de falhas aditivas; e *MultiplicativeFault* e *RandomMultiplicativeFault* para injeção de falhas multiplicativas.
- *Connectors* – Contém componentes especiais chamados "conectores" que permitem interligar modelos de tipos diferentes. Conectores recebem um determinado tipo de sinal como entrada (como por exemplo, um valor real) e produz como saída, um sinal de outro tipo (um sinal booleano, por exemplo).
- *Util* – Pacote com componentes e funções de utilidade geral para construção de modelos.
- *Examples* – Contém exemplos de modelos de detecção de falhas para alguns sistemas.

6.3 Organização do sistema

O sistema Auto-FDI foi organizado em pacotes de software, definidos por elementos inter-relacionados. De acordo com [Fowler1997], a organização de um sistema de médio a grande porte em pacotes é um dos instrumentos mais importante para controlar a estrutura do sistema, além de facilitar o entendimento do mesmo.

O agrupamento dos elementos em pacotes foi realizado observando-se os seguintes princípios de projeto de pacotes [Fowler1997]:

- *Common Closure Principle* (CCP): As classes em um pacote deveriam estar próximas em relação a alterações, ou seja, uma alteração que afeta o pacote possivelmente afeta todas as classes neste pacote.

- *Common Reuse Principle* (CRP): As classes em um pacote são reusadas em conjunto. Se uma classe do pacote é reusada, é possível que outras classes do pacote também sejam reusadas.

Esta seção tem como objetivo apresentar a estrutura de pacotes do Auto-FDI. A apresentação é "top-down", ou seja, primeiro são apresentados os pacotes no nível superior, e em seguida os sub-pacotes e classes no nível mais baixo da hierarquia.

A Figura 40 mostra os principais pacotes do sistema. Uma breve descrição de cada um desses pacotes é fornecida na Tabela 3.

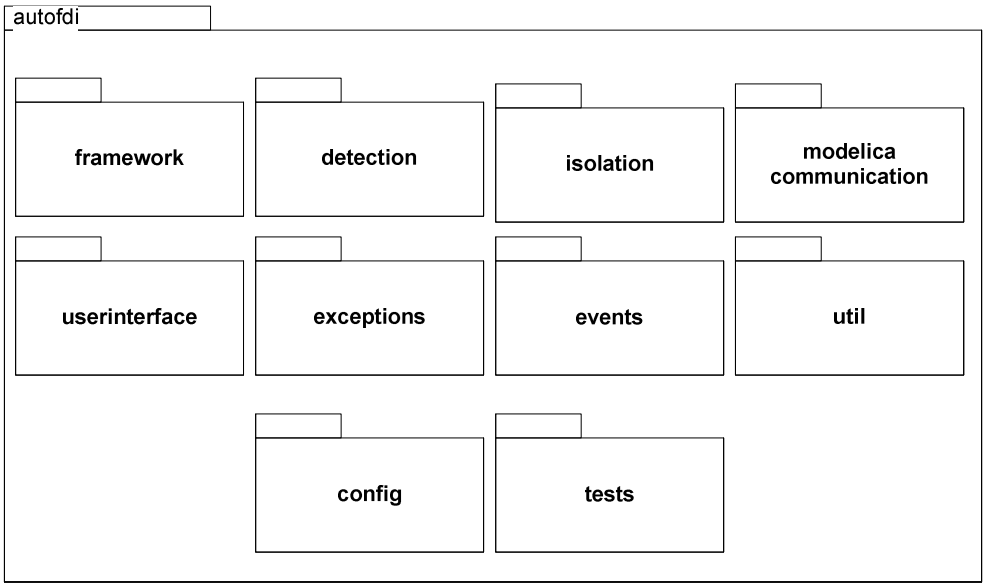


Figura 40: Principais pacotes da ferramenta Auto-FDI

Pacote	**Descrição**
framework	Define um conjunto de interfaces e classes abstratas para o domínio de problema de diagnóstico de falhas em geral.
detection	Implementa a parte de detecção de falhas do processo de detecção e diagnóstico de falhas.
isolation	Implementa a parte de isolamento de falhas do processo de detecção e diagnóstico de falhas.
modelicacommunication	Implementa a comunicação do sistema Auto-FDI com a ferramenta de modelagem Modelica.

userinterface	Implementa a interface gráfica da ferramenta, que inclui o *Auto-FDI Editor* e o *Auto-FDI Executor*.
exceptions	Contém as classes de exceções do sistema Auto-FDI que não foram implementadas no pacote framework.
events	Contém as classes de eventos do sistema Auto-FDI que não foram implementadas no pacote framework.
util	Contém classes utilitárias para todas as outras classes do sistema Auto-FDI.
config	Realiza a configuração da ferramenta. Também contém as classes responsáveis por instanciar, conectar e configurar os componentes de software da ferramenta.
tests	Testa os componentes responsáveis pelo funcionamento da ferramenta, através de testes de unidades e testes de aceitação.

Tabela 3: Descrição dos principais pacotes da ferramenta Auto-FDI

Nas subseções que seguem, os pacotes mais relevantes do sistema são descritos, bem como as classes e interfaces principais que compõem cada pacote. A seguinte convenção de nomes é adotada: nomes de interfaces começam com a letra "I" e nomes de classes abstratas começam com a palavra "Abstract".

6.3.1 *O pacote framework*

Este pacote contém um conjunto de interfaces e classes abstratas que podem ser usadas para a implementação de soluções de diagnóstico de falhas em geral. Ele provê a funcionalidade abstrata comum a este domínio de problema.

A implementação deste pacote foi baseada, em parte, em um outro trabalho, intitulado "Um framework para auxílio na criação de diagnosticadores de falhas em sistemas", cuja referência é [Pontes2008]. Ele foi projetado para ser o mais extensível e flexível possível, permitindo o reuso em outras aplicações de diagnóstico.

Neste pacote estão inclusas as interfaces que definem a arquitetura do Auto-FDI e que foram descritas no capítulo 5. Dirija-se àquele capítulo para obter a descrição de cada interface.

6.3.2 O pacote *isolation*

Este pacote contém as classes que implementam o isolamento de falhas do processo de detecção e diagnóstico de falhas. Foram implementados dois métodos de isolamento de falhas: o isolamento usando matriz binária de diagnóstico (*Binary Diagnostic Matrix* –BDM, em inglês) e o isolamento automático usando uma técnica baseada em grafo. O diagrama com as principais classes do pacote é mostrado na Figura 41.

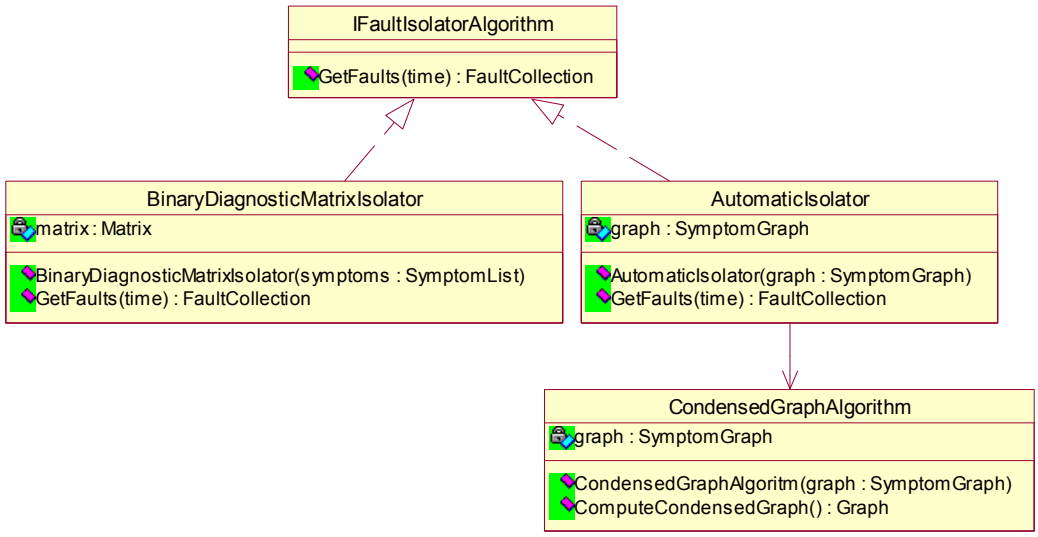


Figura 41: Diagrama de classes do pacote *isolation*

A classe *BinaryDiagnosticMatrixIsolator* foi desenvolvida durante o trabalho de [Pontes2008]. Tal classe implementa o método de isolamento de falhas baseado em uma matriz, chamada matriz binária de diagnóstico, que contém a relação entre os sintomas e as possíveis falhas no sistema. Os sintomas podem ser gerados durante o processo de detecção de falhas ou podem ser sintomas heurísticos, obtidos através da observação do sistema por especialistas. Esse método tem a grande desvantagem de permitir apenas o isolamento de uma única falha (*single fault diagnosis problem)* em que é considerada a hipótese de que não ocorre mais de uma falha no sistema ao mesmo tempo. A teoria por trás desse método e os detalhes de implementação são apresentados em [Pontes2008] (da mesma autora).

A classe *AutomaticIsolator* implementa o técnica de isolamento de falhas abordada no capítulo 3. A classe possui um grafo (*SymptomGraph*) que representa o

modelo de propagação de falhas do sistema, mas precisamente, o modelo de propagação de falhas descrito na seção 3.2. A classe *CondensedGraphAlgoritm* é responsável por realizar a redução do grafo, conforme descrito na seção 0 do mesmo capítulo.

6.3.3 O pacote *detection*

Pacote definido para agrupar as classes que implementam detecção de falhas do processo de detecção e diagnóstico.

Vale ressaltar que optamos por implementar o processo de detecção de falhas do Auto-FDI em *Modelica*, devido aos motivos já expostos no capítulo 5. Em virtude disto, existe atualmente, apenas uma classe neste pacote - a classe *ModelicaFaultDetection,* ilustrada no diagrama da Figura 42. Esta classe é responsável por obter o resultado da execução do modelo *Modelo de Detecção*, através de chamadas aos métodos definidos na interface *IModelicaAdapter* (pertencente ao pacote *modelicacommunication*).

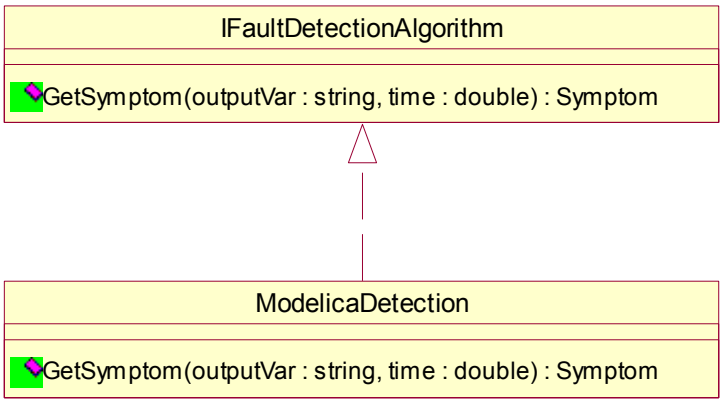


Figura 42: Diagrama de classes do pacote detection

6.3.4 O pacote *modelicacommunication*

Este pacote realiza a comunicação entre o sistema Auto-FDI e o software de execução do modelo *Modelica*. As classes que compõe este pacote são listadas no diagrama abaixo.

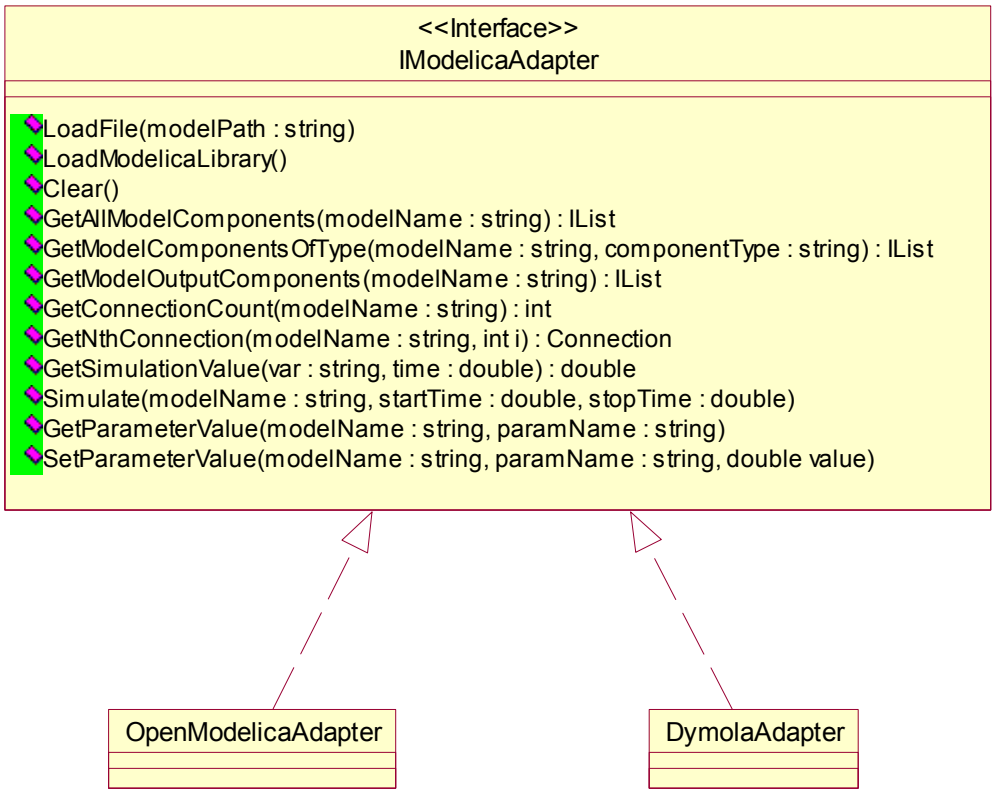


Figura 43: Diagrama de classes do pacote modelicacommunication

A interface *IModelicaApdater* funciona como uma ponte entre o sistema Auto-FDI e o software de execução do modelo *Modelica* permitindo fácil acesso à informações sobre o modelo *Modelica* (estrutura do modelo do sistema, as variáveis, parâmetros, etc.) e ao resultado da simulação do modelo. As classes *OpenModleicaAdapter* e *DymolaAdapter* implementam esta interface e obtém essas informações através das ferramentas *OpenModelica* e *Dymola* respectivamente.

7 Verificação e Validação

Capítulo 7

Neste capítulo apresentamos a metodologia utilizada para a verificação e validação do Auto-FDI. Verificação e Validação (V & V) de software é o processo que visa garantir que o software desenvolvido atende a sua especificação (verificação) e que ele satisfaz as expectativas do cliente (validação) [Sommerville2004]. Em outras palavras, verificação envolve constatar que o software está sendo desenvolvido de acordo com os requisitos funcionais e não funcionais especificados, enquanto que validação tem como finalidade mostrar que o software faz realmente o que o cliente espera. Técnicas de verificação e validação incluem: revisões de requisitos, revisões de projeto, inspeções de código e testes, avaliação de usuários, entre outras.

Embora as técnicas de inspeções e revisões tenham sido usadas durante o desenvolvimento do Auto-FDI, a verificação da ferramenta foi caracterizada principalmente por testes de software. A metodologia de testes adotada é descrita na seção 7.1. A validação da ferramenta, por sua vez, foi feita através da realização de um estudo de caso por potenciais usuários da ferramenta. A validação teve como principal objetivo avaliar a satisfação dos usuários em relação à utilidade da ferramenta. Não é objetivo deste trabalho fazer uma validação rigorosa da ferramenta. No entanto, uma situação de uso real da ferramenta demonstrou sua aplicabilidade e permitiu com que melhorias fossem sugeridas. A seção 7.2 apresenta a metodologia de validação adotada e os resultados obtidos.

7.1 Metodologia de testes

A atividade de teste tem uma importância fundamental no desenvolvimento de software. O uso de uma boa metodologia de testes permite reduzir os defeitos (*bugs*) apresentados pelo software, aumentando consideravelmente a qualidade e confiabilidade do produto desenvolvido. Além disso, permite reduzir os custos de desenvolvimento – pois possibilita encontrar e corrigir erros mais rapidamente; e de manutenção – permitindo modificar (ou refatorar) o código sem adição de novos defeitos.

A metodologia de testes adotada no desenvolvimento do Auto-FDI leva em consideração os princípios básicos relacionados à atividade de testes. Vários aspectos foram considerados, entre os quais destacamos:

- Os testes devem rastrear os requisitos - Testes é uma forma de verificar se as funcionalidades do software foram implementadas corretamente;
- Testes devem ser automatizados – A execução dos testes deve ser realizada freqüentemente, e, portanto deve ser uma atividade realizada com pouca quantidade de tempo e esforço;
- Uso de ferramentas – Devem ser usadas ferramentas que facilitem a criação e execução, bem como a verificação da cobertura dos testes;

O Auto-FDI foi desenvolvido usando o "Desenvolvimento Guiado por Testes" (*Test Driven Development* – TDD) [Beck2002]. Foram usados dois tipos de testes: testes de unidade e testes de aceitação. Ambos foram implementados usando o framework *NUnit* [NUnit]. A infraestrutura de testes também incluiu análise de cobertura de código, realizada com o auxílio da ferramenta *NCover* [NCover].

7.2 Validação da ferramenta

A validação da ferramenta foi feita através da realização de um estudo de caso por potenciais usuários e teve como principal objetivo avaliar a satisfação dos usuários em relação à utilidade da ferramenta. Participaram desta avaliação o engenheiro da CHESF Iony Patriota (cliente da ferramenta) e os professores da Universidade Federal de Campina Grande, Jacques Philippe Sauvé e Jorge Cesar Abrantes de Figueiredo. O estudo de caso, na forma de um manual de uso da ferramenta encontra-se no Apêndice C desta dissertação.

A execução do estudo por parte do cliente foi particularmente importante para avaliar se os requisitos estão realmente de acordo com suas expectativas. Através da utilização da ferramenta o cliente pode ter uma maior clareza do que é necessário que a ferramenta faça para satisfazê-lo plenamente. Além disso, permitiu com que ele pudesse contribuir com sugestões de melhorias.

A seguir mostramos a satisfação dos principais requisitos e alguns resultados práticos de uso da ferramenta.

7.2.1 Satisfação dos requisitos

Esta seção contém uma breve discussão sobre a satisfação dos requisitos mais relevantes da ferramenta.

RF 1. [Criação de Modelos]: Deve existir uma interface gráfica que permita criar modelos visualmente, através da composição de componentes de uma biblioteca de sub-modelos e funções. Deve ser possível criar os seguintes tipos de modelos:

- Modelos lógicos (booleano)
- Modelos contínuo estático, representado por equações algébricas;
- Modelo contínuo dinâmico, representado por equações diferenciais;
- Modelo de eventos discretos;
- Modelo probabilístico;
- Modelo representado por redes de Petri;
- Modelos representados por máquina de estados finitos;

Comentários:

Ao invés de "reinventar a roda", o Auto-FDI utilizou o *SimForge* como ferramenta de criação e execução de modelos. Conforme já discutido, o *SimForge* permite criar os modelos visualmente, através da composição de componentes (veja seção 6.1.1). Os modelos são descritos usando a linguagem *Modelica,* que permite a criação de modelos usando vários tipos de formalismo, incluindo os tipos de modelos elencados no requisito (veja seção 6.2).

Embora não esteja incluso no estudo de caso a criação de todos os tipos de modelos listados no requisito, eles foram criados ao longo do desenvolvimento da ferramenta e estão disponibilizados em sua biblioteca de componentes (no pacote *Diagnostics.Examples*). Também foram disponibilizadas várias bibliotecas para facilitar a criação desses modelos (veja seção 6.2).

Os usuários da ferramenta se mostraram extremamente satisfeitos com a implementação deste requisito, principalmente por adotar uma linguagem de modelagem padronizada, que é a linguagem *Modelica*. Também foi bem aceito o fato de usar uma ferramenta de código aberto para a criação dos modelos, o *SimForge*.

RF 2. [Interligar modelos de tipos diferentes] Deve ser possível criar modelos interligando sub-modelos de tipos diferentes (como por exemplo, criar um modelo interligando um modelo lógico com um modelo contínuo).

Comentários:

Esse requisito foi atendido através do uso de componentes especiais, chamados "conectores" (veja decisão arquitetural 11, no capítulo 5). Foram disponibilizados vários conectores na biblioteca de componentes do Auto-FDI (pacote *Diagnostics.Connectors*). Embora não tenham sido criados conectores para interligar todos os tipos de modelos elencados no requisito "[Criação de Modelos]", é perfeitamente possível que novos conectores sejam criados para este fim.

RF 4. [Adição de modelos] Deve ser possível adicionar facilmente na biblioteca de modelos da ferramenta, novos modelos criados pelo usuário;

Comentários:

Novos componentes podem ser criados facilmente e adicionados na biblioteca de componentes da ferramenta através do *SimForge* (veja seção 6.1.1).

RF 5. [Verificação de consistência de modelos] A ferramenta deve permitir que o usuário valide o modelo criado, ou seja, verifique se restrições estruturais estão sendo observadas, impedindo a criação de composições inválidas (por exemplo, conectando sinais de tipos incompatíveis).

Comentários:

A verificação de inconsistências é feita pela própria ferramenta de modelagem, durante a criação do modelo. A tela abaixo ilustra a indicação de inconsistência durante a criação de um modelo, onde a saída do componente *const* (tipo real) é incompatível com a entrada do componente *booleanToReal* (tipo booleano).

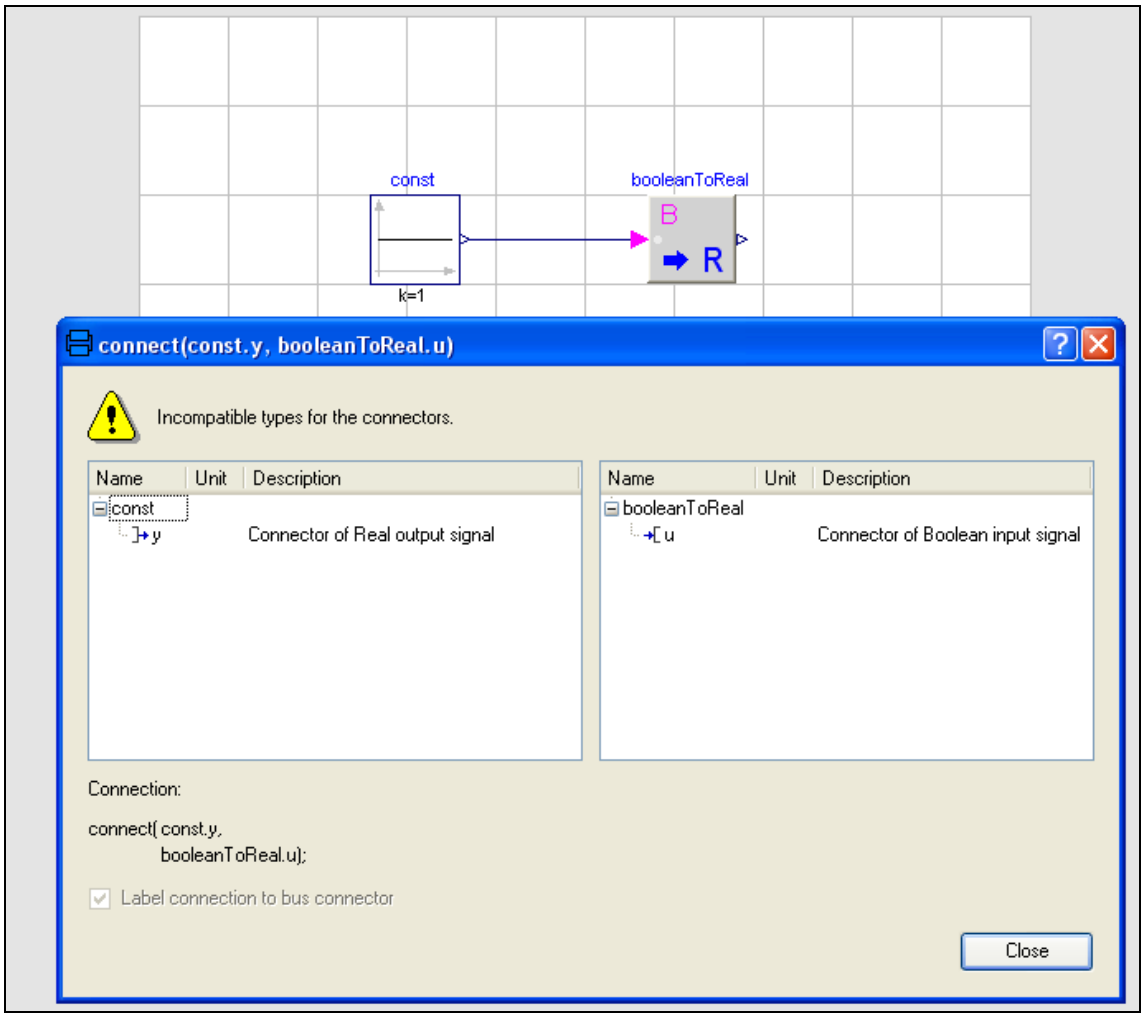


Figura 44: Verificação de inconsistências no modelo

RF 6. [Execução de modelos] A ferramenta dever permitir a execução do modelo através da injeção de sinais de entrada e observação de sinais de saída. Para executar um modelo, deve ser possível fornecer valores para os parâmetros. A execução de modelos pode se apoiar em ferramentas especializadas (como por exemplo, *SimForge, Dymola*).

Comentários:

A execução de modelos é feita no *SimForge*, embora possa usar outra ferramenta que dê suporte a linguagem *Modelica* (como por exemplo, *Dymola* ou *MathModelica*). Para a injeção de sinais de entrada no modelo são usados componentes *Modelica* especiais chamados "adaptadores" (veja decisões arquiteturais 9 e 10, no capítulo 5). A seção 1.4 do manual da ferramenta (Apêndice C) ilustra o uso de adaptadores no estudo

de caso. Os parâmetros do modelo pode ser alterados no *SimForge*, no *Auto-FDI Editor* e no *Auto-FDI Executor*.

Os sinais de saída são observados através de gráficos no *SimForge* ou, alternativamente, no *Auto-FDI Executor* durante a realização do diagnóstico.

RF 7. [Geração de sinais] A ferramenta deve possuir componentes que gerem sinais. Tais sinais podem ser utilizados como entradas para modelos ou em qualquer outro ponto em que um sinal seja necessário para alimentar uma parte de um modelo. Deve ser possível gerar sinais por conteúdo especificado em arquivo, através de funções booleanas, funções contínuas algébricas e seqüências de eventos.

Comentários:

O Auto-FDI disponibiliza várias bibliotecas de componentes para criação de modelos (veja seção 6.2). Entre estes componentes, existe uma grande variedade de componentes que geram sinais. Para gerar sinais através de um conteúdo especificado de arquivo, temos o componente *FileReader*, que encontra-se no pacote *Diagnostics*. Um exemplo de utilização desse componente pode ser encontrado na seção 1.4 do manual (Apêndice C). Além do *FileReader*, a *Biblioteca Padrão Modelica* contém várias funções e componentes para leitura de arquivos (no pacote *Modelica.Utilities*) e que podem ser usados para criação de novos componentes de geração de sinais.

Existe também na *Biblioteca Padrão Modelica* uma grande variedade de componentes para geração de sinais através de diversos tipos de funções. Por exemplo, no pacote *Modelica.Blocks.Sources* temos componentes de geração de sinais através de funções contínuas (ex.: *BooleanConstant*, *BooleanStep* e *BooleanPulse*), de funções contínuas (ex.:*Constant*, *Sin*, *Ramp*), de seqüências de eventos (ex.: *TimeTable* e *CombiTimeTable*), entre outros.

Enfim, ao adotar a linguagem *Modelica* para descrição dos modelos, incorporamos ao Auto-FDI uma imensa gama de componentes de geração de sinais, fazendo com que este requisito fosse facilmente atendido.

RF 8. [Criação do ambiente de diagnóstico] A ferramenta deve possuir um editor para a criação do ambiente de diagnóstico – a interface que será usada pelo operador para realizar o diagnóstico de falhas no sistema. Deve ser possível criar uma interface personalizada de acordo com o sistema a ser avaliado. A interface criada poderá conter imagens representando o sistema a ser avaliado, figuras, links, gráficos, etc.;

Comentários:

Para atender a este requisito foi desenvolvido o *Auto-FDI Editor* (veja seção 6.1.2). Com ele é possível criar interfaces altamente personalizadas para o sistema a ser avaliado. Uma barra de ferramenta contém componentes de figuras, links, gráficos, *labels* e alarmes, que podem ser arrastados para uma área de montagem da interface.

De um modo geral, os usuários mostraram-se bastante satisfeitos com o editor para criação do ambiente de diagnóstico. Apenas algumas pequenas sugestões foram sugeridas por parte do cliente, como por exemplo, a possibilidade de escolher vários tipos de gráficos (gráficos de barras, pizza, etc.) para visualizar os dados.

RF 9. [Gráficos] O ambiente de diagnóstico deve permitir a geração de gráficos para visualização do processo de diagnóstico (valores de entradas e saídas do sistema, saída do modelo, resíduos, sintomas, falhas, etc.).

Comentários:

Durante a criação do ambiente de diagnóstico no Auto-FDI *Editor,* é possível adicionar gráficos que serão plotados durante a realização do diagnóstico no Auto-FDI *Executor* (veja seção 1.5 do manual no Apêndice C). Ao adicionar um gráfico, o usuário define as variáveis que serão exibidas no mesmo (valores entradas e saídas do sistema, resíduos, sintomas, falhas, etc.). Os gráficos são plotados durante a realização do diagnóstico no Auto-FDI *Executor*.

Sendo assim, o requisito também foi atendido. No entanto, conforme já dito, foi sugerido que pudesse ser escolhido entre diversos tipos de gráficos (gráficos de barras, pizza, etc.).

RF 12. [Modelo de sistema e modelo de sinal] Deve ser possível criar soluções de diagnóstico usando métodos de detecção de falha baseados: (a) em modelos de sistemas; (b) em modelos de sinais; (c) uma combinação de ambos.

Comentários:

O estudo de caso apresentado no apêndice C demonstra a criação de uma solução de diagnóstico baseada em modelo do sistema. Na solução, é feita uma comparação entre os valores das variáveis do sistema com valores esperados (obtidos do modelo), gerando assim os resíduos. Depois é feita a avaliação de resíduos, onde são gerados os sintomas. Consulte a seção 2.3 para obter mais detalhes sobre o processo de detecção de falhas baseada em modelos do sistema. A Figura 45 mostra o Modelo de Detecção para o sistema de excitação.

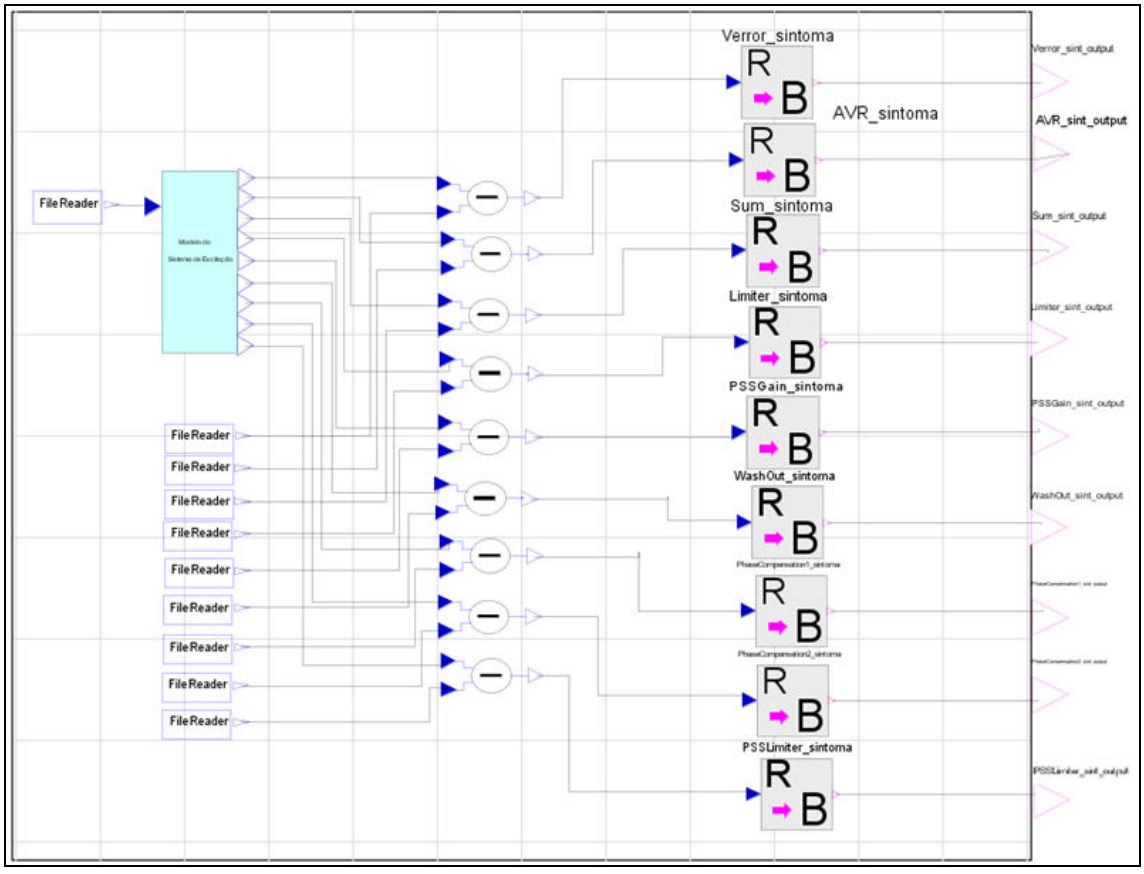


Figura 45: Modelo de Detecção para o sistema de excitação

Durante o desenvolvimento do Auto-FDI, também foram feitos exemplos utilizando modelos de sinais e exemplos que misturam modelos de sistemas com modelos de sinais. Componentes que implementam análise de sinais, a exemplo da análise de *Fourier*, estão disponíveis na biblioteca *Diagnostics.Signal*. Neste pacote também estão inclusos componentes especiais, chamados *componentes de ligação*, que

são usados para especificar as relações de dependência entre os componentes do sistema em soluções que utilizam modelos de sinais, com o objetivo de tornar possível o isolamento de falhas. Um componente de ligação recebe um sinal como entrada e, simplesmente, propaga este sinal para sua saída. Se for uma componente folha do sistema (a saída do componente não é entrada para outro componente do sistema), o componente de ligação simplesmente ignora o sinal de entrada. O modelo ilustrado na Figura 46 exemplifica o uso de componentes de ligação, onde o componente comp1 é um modelo do sistema e os componentes comp2, comp3 e comp4 são componentes de ligação.

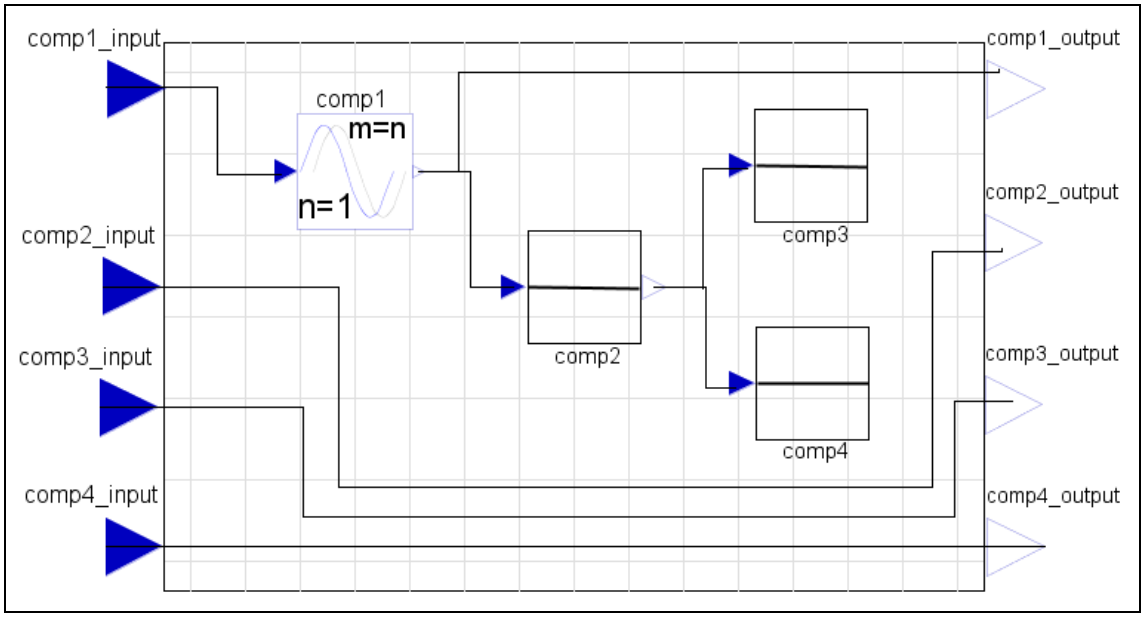


Figura 46: Modelo com componentes de ligação

Esse modelo foi criado para demonstrar uma solução de diagnóstico usando a combinação de técnicas baseadas em sinais com técnicas baseadas em modelos de sistemas. O modelo de detecção ilustrado na figura abaixo usa esse modelo para exemplificar o uso de ambas as técnicas de detecção de falhas.

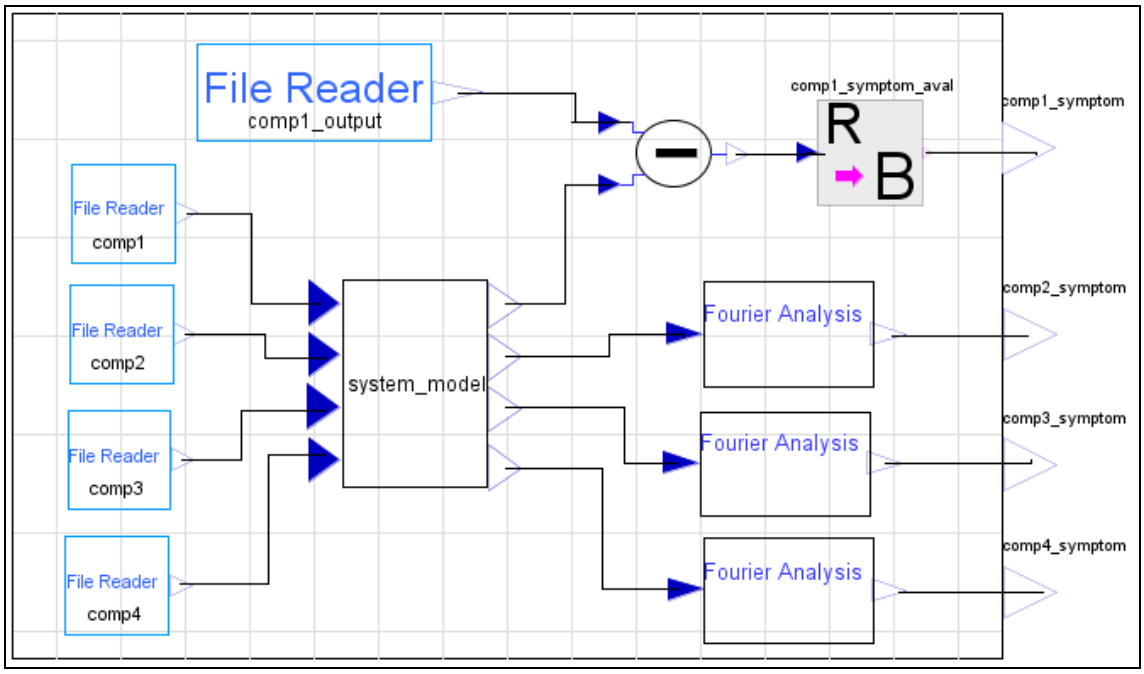


Figura 47: Detecção de falhas usando técnicas baseadas em modelo do sistema e modelo de sinais

Os exemplos acima apresentados comprovam a satisfação deste requisito.

n

RF 17. **RF 13. [Realização do diagnóstico]** Deve ser possível realizar o diagnóstico através do ambiente de diagnóstico criado. Valores para parâmetros e para os sinais de entrada e saída oriundos de um sistema real devem ser informados. O diagnóstico deve indicar as possíveis falhas no sistema sendo avaliado. O isolamento de falhas deve ser realizado de forma automática, dado o modelo estrutural do sistema, sem necessidade de implementação de algoritmos especiais para diferentes sistemas. Falhas devem ser identificadas de forma mais precisa possível, dado o nível de detalhes do modelo. Deve ser possível realizar o diagnóstico em sistemas com até 1000 componentes em no máximo 10 segundos.

Comentários:

Para atender esse requisito foi desenvolvido o Auto-FDI *Executor* (veja seção 6.1.3). O Executor permite a visualização e/ou alteração dos parâmetros do modelo que foram selecionados durante a montagem na interface no Auto-FDI *Editor*. Os sinais de entrada e saída do sistema são obtidos de arquivos (ou outra fonte, dependendo dos adaptadores de entrada usados). Os arquivos são informados através dos parâmetros dos componentes adaptadores (veja Figura 73 no Apêndice C).

A indicação de falhas é feita de duas maneiras: através dos alarmes e na ***janela de exibição de falhas***. Para mais detalhes, veja a seção 1.6 do manual no Apêndice C.

O isolamento de falhas é feito através da técnica introduzida no capítulo 3, denomiada *isolamento automático*. A técnica considera a propagação de falhas e a falta de informação sobre determinados componentes do sistema avaliado. No Apêndice D são expostos alguns cenários de configuração de sistemas com o respectivo resultado do isolamento de falhas. Vários testes de unidade e de aceitação foram feitos para assegurar que o isolamento automático funciona conforme o esperado. Os testes incluem, além de outros, todos os cenários expostos no apêndice D.

Para validação desse requisito também foi realizada uma avaliação de desempenho do isolamento automático. Os resultados dessa avaliação encontram-se na seção 7.2.2.

RF 15. [Simulação do diagnóstico] A ferramenta deve permitir a simulação do diagnóstico para fins de validação. Deve ser possível injetar falhas no sistema durante a simulação.

Comentários:

Para fins de validação do diagnóstico de falhas, é possível incluir no Modelo de Detecção, em substituição aos adaptadores de fonte de entrada do sistema, um modelo do sistema com componentes de injeção de falhas. Na biblioteca de componentes do Auto-FDI são disponibilizados componentes tanto para injeção de falhas aditivas quanto falhas multiplicativas (no pacote *Diagnostics.FaultInjection*). Os parâmetros dos componentes de injeção de falhas são alterados de mesma maneira que é feito para outro componente do modelo *Modelica*.

O estudo de caso no Apêndice C inclui uma seção sobre a simulação do processo de detecção e diagnóstico de falhas (seção 1.7). Os usuários da ferramenta se mostraram satisfeitos com a implementação deste requisito.

RNF 3. [Arquitetura extensível] A ferramenta deve possuir uma arquitetura que permita ser estendida facilmente com a adição de novos componentes

funcionais. A adição de componentes poderá ser feita por terceiros, sem necessidade de recompilação de código.

Comentários:

Para atender ao requisito de extensibilidade, o Auto-FDI foi centrado em uma arquitetura baseada em componentes (veja *decisão arquitetural* 1, no capítulo 5). Uma arquitetura baseada em componentes permite a adição de novos componentes com facilidade sem a necessidade de recompilação do código, o que torna a solução bastante flexível.

7.2.2 Avaliação de desempenho do isolamento automático de falhas

A avaliação de desempenho do isolamento automático foi feita com base em estudos experimentais. Durante a realização dos experimentos utilizou-se um computador pessoal com processador Intel® Core™2 Duo E4600 @ 3.00 GHz com 2GB de memória RAM.

Os experimentos foram feitos através da execução do algoritmo de isolamento automático em modelos de propagação de falhas gerados randomicamente (veja seção 3.2). Foram criados dois cenários para os experimentos, o primeiro com modelos de propagação de falhas esparsos e o segundo com modelos de propagação de falhas densos[1]. Os modelos esparsos foram gerados com uma quantidade de arestas do grafo igual a cinco vezes o número de vértices. Já os modelos densos foram gerados com o número máximo de arestas no grafo, ou seja, $e = n\ (n - 1)$, onde n e e são, respectivamente, o número de vértices e de arestas do grafo. Em ambos os cenários foram gerados modelos de propagação de falhas variando-se o número de componentes do modelo, iniciando em 100, chegando até 1000 (com incremento de 100). Os modelos foram gerados com 30% de probabilidade de uma saída de um componente apresentar sintoma de falha e 30% de probabilidade de uma saída um componente apresentar estado indeterminado.

[1] Seja $G=(V, E)$ um grafo, onde $|V|$ é o número de vértices e $|E|$ o número de arestas do grafo. G é um *grafo esparso* se $|E| = O(|V|)$ enquanto que G é um *grafo denso* se $|E| = O(|V|^2)$ [Preiss98].

Para ambos os cenários, o isolamento automático foi realizado com desempenho satisfatório. Para modelos esparsos, o isolamento para até 1000 componentes foi feito em menos de 100ms, tempo bem abaixo do tempo máximo exigido como requisito (10 segundos).

Em modelos densos, o isolamento para até 1000 componentes foi feito em tempo médio de menos de 3 segundos (média para 5 medições), conforme mostra o gráfico da Figura 48. Vale salientar que os resultados para este cenários foram obtidos com grafos contendo o número máximo de conexões entre os componentes. Este cenário, embora não seja realista, permite avaliar o desempenho do algoritmo no pior caso possível.

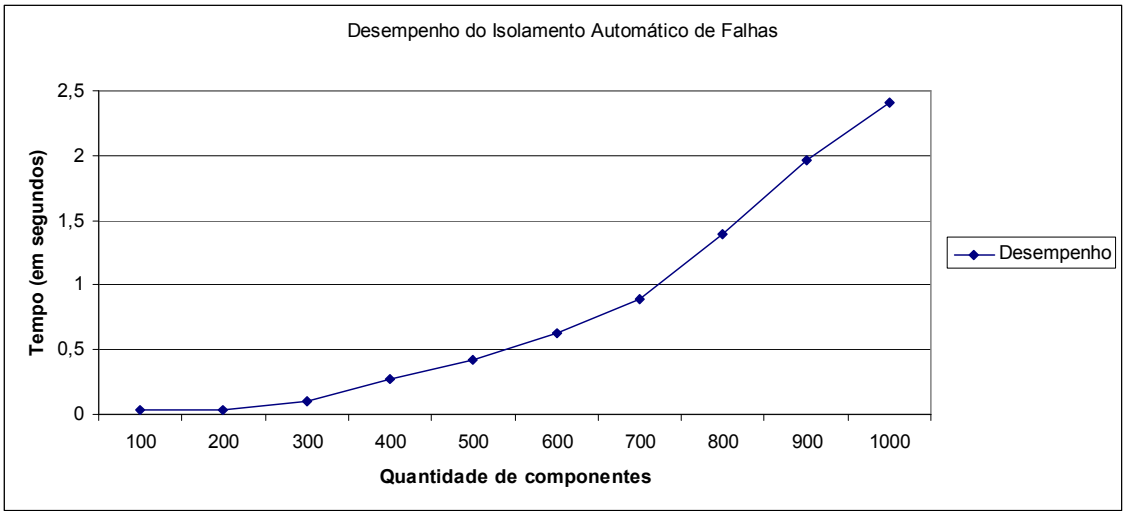


Figura 48: Desempenho do isolamento automático para grafos densos

8 Conclusão e Sugestões para Trabalhos Futuros

Capítulo 8

Neste trabalho, apresentamos uma ferramenta de detecção e isolamento de falhas em sistemas, chamada Auto-FDI. A ferramenta utiliza técnicas baseadas em modelos e possui várias características e funcionalidades que não são contempladas pelas ferramentas de detecção e diagnóstico existentes.

O principal objetivo do desenvolvimento da ferramenta foi o de prover uma prova de conceito para a técnica de isolamento de falhas introduzida no capítulo 3, denominada de *isolamento automático*. O isolamento automático constitui uma contribuição deste trabalho no tocante ao desenvolvimento de técnicas e algoritmos de isolamento de falhas. A técnica desenvolvida considera a propagação de falhas e a falta de informação sobre alguns componentes do sistema e pode ser aplicada em diferentes setores da indústria.

Para concretização deste trabalho foi inicialmente realizada uma revisão bibliográfica do estado da arte em técnicas de detecção e diagnóstico de falhas, com o intuito de compreender o objetivo e aplicabilidade das abordagens existentes. Esse estudo resultou na escolha de uma abordagem baseada em grafos para o isolamento de falhas, devido principalmente a sua grande aplicabilidade em diversos contextos.

Também foram feitos um levantamento e posterior análise de ferramentas de detecção e diagnóstico de falhas existentes na literatura. Através dessa análise, constatamos que as ferramentas existentes não contemplam várias características e funcionalidades bastante relevantes. Diante disto, surgiu a motivação para a criação de uma nova ferramenta com o intuito de preencher algumas dessas lacunas deixadas e, ainda, comprovar a aplicabilidade da técnica de isolamento de falhas desenvolvida.

Ao longo da dissertação foi abordado todo o processo de especificação, projeto e implementação da ferramenta proposta. Os requisitos foram obtidos a partir de conversas com os diversos *stakeholders* da ferramenta e através da análise das ferramentas existentes. A partir de então, foi definida a arquitetura para o sistema,

apresentada no capítulo 5. O projeto detalhado e aspectos de implementação da ferramenta são apresentados no capítulo 6.

O Auto-FDI considerou características essenciais para ferramentas desta natureza. Dentre as principais funcionalidades, destacamos:

- Permite a criação de modelos a partir da montagem de componentes;
- Inclui uma vasta gama de componentes para a criação de modelos para sistemas de diversos setores da indústria. Novos componentes podem ser adicionados facilmente.
- Dá suporte a vários tipos de modelos, entre eles: modelos lógicos (booleano), modelos representados por equações algébricas, modelos representados por equações diferenciais, modelos de eventos discretos, modelos probabilísticos, modelos representados por redes de Petri e modelos representados por máquina de estados finitos.
- Permite interligar modelos de tipos diferentes para composição de um novo modelo.
- Permite usar técnicas de detecção e diagnóstico de falhas baseadas em modelos de sistemas e modelos de sinais;
- Realiza o isolamento de falhas automático, sem a necessidade de implementação de algoritmos adicionais para novos sistemas. Alternativamente, novos métodos de isolamento de falhas podem ser criados e acoplados facilmente à ferramenta;
- Permite a simulação do processo de detecção e diagnóstico de falhas e possui mecanismos de injeção de falhas, para fins de validação do processo de detecção e diagnóstico;
- Possui uma arquitetura baseada em componentes, permitindo adicionar facilmente novos componentes funcionais (como por exemplo, componentes de detecção e diagnóstico de falhas, de injeção de sinais de entrada, etc.), sem necessidade de recompilação do código;
- Permite a integração com ferramentas especializadas em modelagem.

A validação da ferramenta foi feita através da realização de um estudo de caso por potenciais usuários e teve como principal objetivo avaliar a satisfação dos usuários em relação a sua utilidade. Através de uma situação de uso real, foi possível comprovar sua aplicabilidade, além de permitir que melhorias fossem sugeridas. O resultado da satisfação dos requisitos encontra-se na seção 7.2.1.

Ademais, a execução do estudo de caso por parte do cliente foi particularmente importante para avaliar se os requisitos estão realmente de acordo com suas expectativas. Através da utilização da ferramenta o cliente pode ter uma maior clareza do que é necessário que a ferramenta faça para satisfazê-lo plenamente.

O resultado final deste trabalho é uma técnica de isolamento de falhas que pode ser aplicada em qualquer sistema para o qual se disponha de seu modelo estrutural e uma poderosa ferramenta de detecção e isolamento de falhas, o que representam contribuições para a área de detecção e diagnóstico de falhas em sistemas. Com o Auto-FDI é possível identificar falhas de forma rápida e confiável, e com isso, melhorar a eficiência do processo de detecção e diagnóstico. Como benefícios diretos, podemos destacar o aumento da disponibilidade, confiabilidade e segurança dos sistemas e a diminuição dos custos de operação e manutenção. A ferramenta também pode ser utilizada para simulação, servindo como uma ferramenta de treinamento dos operadores de sistemas.

Por fim, acreditamos que a ferramenta possa ainda ser utilizada para validação de outros trabalhos área de detecção e diagnóstico de falhas, contribuindo assim para desenvolvimentos na área. Nesse sentido, apresentamos, na seção seguinte, algumas sugestões para trabalhos futuros.

8.1 Sugestões para Trabalhos Futuros

Eis algumas sugestões para trabalhos futuros:

- Implementar outros métodos de detecção e diagnóstico de falhas, a exemplo de métodos baseados em redes neurais e lógica *fuzzy,* que são muito usados nesse domínio de problemas.
- Implementar métodos de identificação de modelos, como por exemplo, *Least Squares* (LS).

- Permitir integração com outras ferramentas de modelagem, como por exemplo, *Simulink* da Matlab.
- Dar suporte à linguagem SCL (*Substation Configuration Description Language*), para permitir detecção e diagnóstico de falhas em sistemas de automação de subestação baseados na norma IEC 61850. IEC 61850 "*Communication networks and systems in substations*" é um padrão internacional para o projeto de automação de subestações que define a comunicação entre os vários dispositivos da subestação e os requisitos do sistema.
- Permitir o uso de modelos não-causais para realização da detecção e diagnóstico de falhas. Esses tipos de modelos não especificam explicitamente as entradas e saídas do sistema, ou seja, não especificam quais variáveis do modelo são de entrada e quais são de saída. Permitir o uso apenas de modelos causais constitui uma limitação do presente trabalho. Isso porque o isolamento automático precisa conhecer o sentido do fluxo para construir o modelo de propagação de falhas do sistema.

Apêndice A - Terminologia

Com o intuito de facilitar a leitura e compreensão da dissertação, apresentamos neste apêndice, a terminologia utilizada nesta monografia. Consultando a literatura, constata-se que a terminologia utilizada na área não é única e muitas vezes inconsistente. Grande parte da terminologia apresentada aqui resulta do esforço do Comitê Técnico SAFEPROCESS em encontrar uma terminologia que seja comumente aceita. O resultado desse esforço foi publicado em [Isermann e Ballé]. Outras definições podem ser encontradas em [Gertler] [Simani].

1. Estados e Sinais

Falha: Um desvio não permitido, de pelo menos uma propriedade característica ou de uma variável do sistema, de seu comportamento aceitável, usual ou padrão.

Distúrbio: Uma entrada adicional e desconhecida atuando no sistema.

Ruído: Uma entrada adicional e desconhecida atuando no sistema, com comportamento aleatório e com média zero.

Resíduo: Um indicador de falha, baseado na diferença entre os valores medidos e valores obtidos de um modelo.

Sintoma: A diferença de uma quantidade observável de um comportamento normal.

2. Funções

Detecção de Falha: Indicação da presença de falha(s) e o instante de detecção.

Isolamento de Falha: Determinação da localização da(s) falha(s). É realizado depois da detecção de falha.

Identificação de Falha: Determinação do tamanho e tipo de uma falha. É realizado depois do isolamento de falha.

Diagnóstico de Falha: Determinação do tipo, tamanho, localização e instante de detecção de uma falha. Vem depois da detecção de falhas. Inclui isolamento e identificação de falha.

Monitorar: Tarefa contínua realizada em tempo real com o propósito de determinar as condições de um sistema, através do registro de informação, e do reconhecimento e indicação de anomalias em seu comportamento.

Supervisionar: Monitorar um sistema e tomar ações apropriadas para manter a operação no caso de falhas.

Proteção: Meios pelo qual um comportamento potencialmente perigoso de um sistema é suprimido caso seja possível; ou meios pelo quais as conseqüências de um comportamento perigoso são evitadas.

3. Modelos

Redundância analítica: Uso de mais de um (não necessariamente idêntico) meio para determinar o valor de variável do sistema, onde um meio usa um modelo matemático na forma analítica.

4. Propriedades do Sistema

Confiabilidade: Habilidade de um sistema executar uma função requerida sob determinadas condições, em um determinado escopo, durante um determinado período de tempo.

Segurança: Habilidade de um sistema não causar perigo a pessoas, equipamentos ou ao ambiente.

Disponibilidade: Probabilidade de um sistema ou equipamento operar satisfatoriamente e efetivamente em qualquer instante de tempo.

Apêndice B - Ferramentas de apóio à detecção e diagnóstico de falhas

Na literatura é possível encontrar alguns trabalhos que descrevem ferramentas de apoio ao diagnóstico de falhas em sistemas. Algumas dessas ferramentas são dedicadas a um setor específico da indústria (como por exemplo, indústria química). Outras, por sua vez, são mais genéricas e permitem que sejam estendidas ou adequadas a diferentes contextos ou áreas de aplicação.

Entre as ferramentas existentes podemos destacar: DIAG [KKKC2004], AMandD (*Advanced and Monitoring Diagnostic Systems*) [KSW2006], *ModBuild* [BSS2006], *FDI-ToolBox* [DAS+2006], *SATOOL* [BL2006], *ControlBuild* [GSHC2006], a toolbox for design of diagnosis systems [FKNA2006], toolbox for model-based fault detection and isolation [TBR+1999] e Um Framework para Auxílio na Criação de Diagnosticadores de Falhas em Sistemas [Pontes2008].

Nas seções seguintes, são detalhadas algumas das ferramentas citadas.

1.1 DIAG

O *DIAG* é uma ferramenta de detecção e isolamento de falhas em sistemas (incluindo atuadores, planta e sensores), que pode ser aplicada em diversos setores da indústria (como por exemplo, indústria química, elétrica, etc.). Ela utiliza informações sobre variáveis do processo e sinais de controles provenientes dos sistemas DCS e SCADA, e prover as seguintes funcionalidades:

- Detecção e isolamento de falhas através de vários métodos;
- Apresentação dos resultados da análise de várias formas: grafos, planilhas, gráficos, diagramas, relatórios e outros;
- Justificativa dos resultados da análise através da apresentação dos sintomas detectados durante processo de detecção e diagnóstico de falhas;
- Armazenamento histórico do diagnóstico de falhas;
- Suporte à tomada de decisões para proteção do sistema monitorado;

- Permite a construção de modelos do processo com redes neurais e lógica fuzzy.
- Permite a exposição dos resultados da análise de falhas.

O *DIAG* permite o uso de várias técnicas de detecção de falhas, incluindo métodos clássicos (verificação de limiar, comparação de sinais redundantes, etc.), além de métodos avançados, baseados em modelos neurais e fuzzy, equações físicas, modelos de processos lineares e modelos heurísticos, ou seja, aqueles baseados na experiência dos operadores.

A ferramenta possui um módulo para a modelagem do sistema com uso de redes neurais e lógica fuzzy dos seguintes tipos: (1) Redes neurais do tipo Perceptron de Múltiplas Camadas; (2) Redes Neuro-Fuzzy; (3) Redes Neuro-Fuzzy Takagi-Sugeno-Kang; (4) Modelos fuzzy Wang-Mendel. Esses modelos são construídos através da análise de dados medidos do sistema durante sua operação normal (sem falha), utilizando variados algoritmos de treinamento.

O isolamento de falhas é realizado através do método F-DTS (descrito em [KKKC2004]), e faz uso de lógica fuzzy para a geração e avaliação de resíduos. As relações entre os sintomas e as falhas são definidas previamente por especialistas humanos e armazenadas na forma de um sistema de informação.

A ferramenta apresenta vários pontos positivos. Entre eles, ressaltamos a grande quantidade de técnicas permitidas para implementação dos métodos de detecção e diagnóstico de falha, possibilitando assim, a escolha do método mais apropriado para cada tipo de sistema. Outro ponto positivo que podemos destacar é robustez aos ruídos e incertezas que possam afetar o sistema (veja robustez, na seção 2.2) A ferramenta permite também a exposição dos resultados da análise de falha para que essa informação possa ser utilizada em outros sistemas (como por exemplo, para visualização nos sistemas DCS ou SCADA). Além disso, sua arquitetura e os métodos de detecção e diagnóstico usados permitem o desenvolvimento contínuo do *DIAG* através da extensão do conhecimento sobre o sistema monitorado.

O *DIAG* é uma ferramenta de DDF poderosa que permite a análise de falhas em sistemas complexos de vários setores da indústria. No entanto, essa ferramenta apresenta algumas desvantagens, dentre as quais destacamos:

- A ferramenta não é facilmente extensível. Embora seja possível estender a ferramenta com a adição de novos métodos de detecção e diagnóstico, a extensão é feita diretamente no código. Alternativamente, uma arquitetura componentizada que permitisse a extensão da ferramenta de forma mais fácil, sem a necessidade de recompilação do código fonte, seria uma estratégia mais adequada.
- A criação de uma solução de diagnóstico utilizando a ferramenta requer um conhecimento profundo do processo de detecção e diagnóstico de falhas. A ferramenta deveria prover meios para guiar o usuário na construção de soluções de diagnóstico (como por exemplo, através de *Wizards*).

1.2 AMandD

O *AMandD* (*Advanced Monitoring and Diagnostic System*) permite a realização de tarefas avançadas de supervisão e diagnóstico em processos industriais complexos de vários setores da industria. O *AMandD* é uma ferramenta do tipo stand alone, possibilitando, porém, a integração com sistemas DSS.

As seguintes funcionalidades estão presentes na ferramenta:

- Modelagem com o uso de métodos de identificação de modelos (como por exemplo, método *Least Squares* - LS) além de métodos baseados modelos teóricos (como por exemplo, modelos com equações de balanço).
- Simulação do modelo criado para verificação da corretude do modelo.
- Simulação da detecção e isolamento de falhas. Essa funcionalidade pode ser usada, por exemplo, para treinamento dos operadores ou otimização do processo de detecção e diagnóstico da organização.
- Determinação (através de estimativas) dos valores de variáveis do sistema quando estas não estão disponíveis (como por exemplo, devido a uma falha de um sensor).
- Detecção e Isolamento de falhas com a utilização de vários métodos.
- Visualização gráfica dos resultados da análise e simulação.

- Envio automático para os operadores (através de e-mail ou mensagens SMS) de informação sobre o estado do sistema e possíveis instruções.

A modelagem do sistema pode ser realizada por meio de equações analíticas ou através de métodos de identificação de modelos via estimativa de parâmetros que utilizam os dados históricos dos processos. A identificação de modelos é realizada com o apoio de w*izards* que guia o usuário passo a passo durante todo o processo. A ferramenta permite a identificação de modelos de vários tipos, a saber: equações de diferenças lineares, equações polinomiais, Fuzzy TSK, modelos de redes neurais.

A detecção de falhas é realizada através de algoritmos que usam Lógica Fuzzy para a avaliação de resíduos. Os conjuntos Fuzzy podem ser determinados automaticamente através de observações estatísticas ou através do conhecimento heurístico dos operadores acerca do sistema.

O isolamento de falhas é realizado através métodos que utilizam conhecimento especialista sobre a relação entre os sintomas e as falhas. Essa relação é descrita através de regras Fuzzy.

A ferramenta possui uma interface gráfica bastante sofisticada e é bastante rica em funcionalidades. Ela implementa vários métodos de detecção e diagnóstico permitindo o design de uma vasta gama de soluções de diagnóstico. Ainda, sua arquitetura é componentizada, o que permite uma fácil extensão de algumas funcionalidades da ferramenta, sem a necessidade de alteração no código. Outro aspecto positivo da ferramenta é a robustez em relação aos ruídos e incertezas que possam afetar o sistema (veja robustez, na seção 2.2).

Como ponto negativo da ferramenta, podemos citar o fato de que ela não expõe suas funcionalidades, permitindo que estas possam ser usadas em outros sistemas. Além disso, ressaltamos que o *AMandD* não permite a integração com outras ferramenta de apóio ao diagnóstico de falhas (como por exemplo, ferramentas especializadas em modelagem).

1.3 ModBuild

O *ModBuild* é uma IDE (*Integrated Development Environment*) de monitoramento que permite criar o modelo de sistemas, validar a consistência desses

modelos e através de uma análise estrutural desses modelos determinar automaticamente algoritmos de detecção e isolamento de falhas para o sistema.

O foco da ferramenta é na criação de modelos do sistema. Ela permite, através de uma interface gráfica, criar facilmente modelos do sistema, arrastando componentes de uma base de componentes e conectando os componentes entre si, formando assim, a estrutura completa do mesmo. Os modelos são representados graficamente através de Diagramas de Processo e Instrumentação (*Process and Instrumentation Diagram* – P&ID, em inglês) e a metodologia de criação de modelos é baseada na modelagem funcional e estrutural do sistema utilizando Grafos de Ligação (Bond Graphs, em inglês). A teoria por trás dessa metodologia é desenvolvida em [OMSD2004].

Uma vez que o modelo estrutural do sistema é criado, o *ModBuild* verifica sua consistência e realiza uma análise estrutural para derivar o modelo dinâmico do sistema (no estado normal e com falhas). A partir do modelo dinâmico do sistema, o *ModBuild* gera os resíduos associados com as Relações de Redundância Analítica (*Analytical Redundancy Relations* – ARRs, em inglês) com base nas restrições do sistema e determina automaticamente os algoritmos de detecção e isolamento de falhas.

Um ponto bastante positivo do *ModBuild* é a maneira de criação de modelos. A ferramenta permite a criação de modelos, através de uma interface gráfica, a partir de montagem de componentes, arrastando e interligando os componentes de uma biblioteca. A ferramenta disponibiliza um número considerável de componentes para construção de modelos do setor da indústria de fluidos térmicos. Além disso, o modelo criado pode ser simulado no ambiente da própria ferramenta ou exportado para simulação no ambiente do *Simulink* da *Matlab*. Outro ponto positivo da ferramenta é que ela determina, automaticamente, a partir do modelo e restrições do sistema, o algoritmo de detecção e isolamento de falhas.

O *ModBuild* reduz significantemente o tempo e custo com modelagem e design de mecanismos de detecção e isolamento de falhas, uma vez que parte dessas tarefas são realizadas de forma automática pela ferramenta. Porém, através de uma análise mais detalhada, constatamos que a ferramenta não implementa métodos avançados de detecção e diagnóstico de falhas (como por exemplos, métodos que considerem perturbações e incertezas de modelagem), reduzindo desse modo, a precisão do diagnóstico (Vide seção 2.2 Atributos desejáveis em um sistema de DDF).

Outro ponto que deve ser ressaltado é que a ferramenta pode ser aplicada somente no setor da indústria de fluídos térmicos, embora a metodologia empregada possa ser aplicada a outros setores da indústria.

1.4 FDI-TOOLBOX

A *FDI-Toolbox* é um conjunto de funções Matlab, desenvolvida pelo *Institute for Automatic Control and Complex Systems* (AKS), com o objetivo de facilitar a construção de soluções de detecção e isolamento de falhas em sistemas. A *FDI-Toolbox* possui funções para auxiliar na construção do modelo do sistema e também no design de métodos de geração e avaliação de resíduos utilizados na detecção e isolamento de falhas.

A construção de uma solução completa de detecção e diagnóstico de falhas usando a *FDI-Toolbox* se dá em três fases: (1) Modelagem do sistema; (2) Geração de resíduos; (3) Avaliação de Resíduos.

A modelagem do sistema é realizada utilizando-se a forma de descrição e esquema de modelagem de uma toolbox da Matlab, a "*Control System Toolbox*". A Control System Toolbox possui ferramentas para analisar, projetar e ajustar sistemas lineares. A *FDI-Toolbox* estende esta toolbox acrescentando funções que levam em consideração as falhas, presença de perturbações e incertezas de modelagem.

Para a geração de resíduos são disponibilizadas funções para a realização de filtragem de sinais e funções que implementam métodos de geração de resíduos via observadores de estado e equações de paridade. Para a avaliação de resíduos as principais funções disponíveis são aquelas para determinação dinâmica de limiares.

A *FDI-Toolbox* se apóia no ambiente de programação Matlab, um software bastante utilizado tanto em instituições de pesquisa quanto na indústria. Com ela é possível utilizar todos os recursos disponibilizados pelo ambiente Matlab e ainda se apoiar nas funções que a *FDI-Toolbox* implementa para facilitar a criação de soluções de detecção e diagnóstico de falhas. Isso torna a FDI-Toolbox uma ferramenta bastante poderosa.

No entanto, devemos salientar que a toolbox não é uma ferramenta destinada a usuário final. Ela não oferece soluções de diagnóstico completas, ela apenas implementa

algumas funções básicas de modo a facilitar o desenvolvimento de soluções de diagnóstico utilizando o ambiente Matlab.

1.5 Um Framework para Auxílio na Criação de Diagnosticadores de Falhas em Sistemas

Em [Pontes2008] é apresentado um framework para a construção de soluções de diagnósticos de falhas baseados em modelos de sistemas, possibilitando mais facilidade e rapidez na construção desses diagnosticadores. A solução apresenta uma arquitetura totalmente componentizada, permitindo reaproveitar os métodos já implementados em vários diagnósticos [Pontes2008]. O framework também faz a comunicação entre os métodos de detecção e os métodos de isolamento e identificação das falhas. A implementação de alguns métodos de detecção e diagnósticos estão disponíveis no framework.

Ferramentas de apóio à detecção e diagnóstico de falhas O trabalho apresentada também o protótipo de uma ferramenta para validação do framework desenvolvido.

1.6 Quadro comparativo

A Tabela 4 apresenta um quadro comparativo das quatro ferramentas descritas neste documento, contemplando as seguintes características:

- C01: Detecção automática de falhas
- C02: Identificação automática de modelos
- C03: Integração com outras ferramentas de modelagem
- C04: Tipos de modelos disponíveis
- C05: Interligação de tipos de modelos diferentes
- C06: Técnicas de detecção de falhas disponíveis
- C07: Técnicas de isolamento de falhas disponíveis
- C08: Extensibilidade
- C09: Usabilidade
- C10: Exposição de funcionalidades
- C11: Criação de modelos baseada em componentes

- C12: Montagem da interface de diagnósticos
- C13: Suporte a modelos de sinais e modelos de sistema
- C14: Simulação do modelo e do processo de diagnóstico

	DIAG	**AMandD**	**ModBuild**	**FDI-Toolbox**
C01	Não	Não	Sim	Não
C02	Sim	Sim	Não	Não
C03	Não	Não	Sim	Não
C04	• Equações físicas • Modelos lineares • Modelos heurísticos • Redes Neurais • Modelos Nebulosos	• Modelos analíticos • Estimativa de parâmetros	• Grafos de Ligação	• Funções de transferência • Espaço de estados
C05	Não	Não	Não	Não
C06	• Métodos clássicos • Redes Neurais • Lógica Nebulosa	• Heurística • Lógica Nebulosa	• ARRs	• Observadores de estado • Equações de paridade
C07	• F-DTS	• Sistemas Especialistas • Regras Nebulosas	• ARRs	• Checagem de Limiares
C08	Baixa	Baixa	Baixa	Baixa
C09	Baixa	Alta	Baixa	Baixa
C10	Sim	Não	Não	Não
C11	Não	Não	Sim	Não
C12	Não	Não	Não	Não
C13	Não	Não	Não	Não
C14	Não	Sim	Não	Não

Tabela 4: Quadro comparativo das ferramentas

Apêndice C – Manual da ferramenta

Este apêndice apresenta uma demonstração do Auto-FDI através de um exemplo completo, que envolve desde a criação de um modelo do sistema até a análise dos resultados do diagnóstico. Este documento não tem como objetivo explorar todas as funcionalidades do Auto-FDI, mas sim, prover um guia prático para uso da ferramenta.

1.1 Visão Geral do Auto-FDI

O Auto-FDI utiliza técnicas baseadas em modelos para realização do diagnóstico de falhas. A idéia básica consiste em comparar sinais obtidos de um sistema com valores esperados, obtidos de um modelo. A diferença entre esses valores, os resíduos, são então usados para identificar possíveis falhas no sistema.

A criação de uma solução de diagnóstico usando o Auto-FDI envolve as seguintes etapas:

1. Criação do modelo que descreve o sistema;
2. Criação do modelo de detecção, que compreende a geração e avaliação de resíduos;
3. Criação do ambiente de diagnóstico, ou seja, a definição da interface gráfica que o operador irá utilizar para realizar o diagnóstico;
4. Execução do diagnóstico e avaliação dos resultados.

Os modelos usados na ferramenta – Não só o modelo que descreve o sistema como também o modelo de detecção de falhas – são descritos usando a linguagem *Modelica*. *Modelica* é uma linguagem de modelagem orientada a objetos formalmente definida que permite a modelagem de sistemas complexos e heterogêneos. É uma linguagem não proprietária e que vem sendo amplamente usada tanto por instituições de pesquisa quanto pela indústria. Junto com a linguagem *Modelica*, a organização *The Modelica Association* também disponibiliza gratuitamente a *Biblioteca Padrão Modelica* (*Modelica Standard Library)*, uma extensa biblioteca com mais de 780

modelos de componentes genéricos e 550 funções multipropósito (referencia à versão 3.0 de fevereiro de 2008). Mais informação sobre a linguagem *Modelica* pode ser encontrada no site: http://www.modelica.org. Além da *Biblioteca Padrão Modelica*, o Auto-FDI disponibiliza a biblioteca *Diagnostics*, uma biblioteca específica para a criação de modelos utilizados no processo de detecção e diagnóstico. Entre os componentes desta biblioteca estão componentes de geração de resíduos, de avaliação de resíduos, de injeção de sinais, de injeção de falhas, etc.

Para a criação desses modelos (etapas 1 e 2), o Auto-FDI vem com um editor gráfico integrado - O *SimForge*. O *SimForge* é um software totalmente grátis e de código aberto. Utiliza o compilador *OpenModelica*, o mesmo compilador *Modelica* usado pelo Auto-FDI. Com o SimForge é possível construir modelos na linguagem *Modelica* de forma rápida e intuitiva, através da montagem de componentes, arrastando e interligando os componentes de uma biblioteca de componentes. O *SimForge* dispõe de toda a Biblioteca Padrão Modelica além de permitir facilmente a extensão de sua biblioteca de componentes. Mais informações podem ser obtidas no site da ferramenta em: http://trac.elet.polimi.it/simforge.

Embora a criação dos modelos possa ser feita usando qualquer outra ferramenta que dê suporte à linguagem *Modelica*, como por exemplo, *Dymola* (http://www.dynasim.se) ou *MathModelica* (http://www.mathcore.com/products/mathmodelica), é importante ressaltar que o Auto-FDI utiliza o *OpenModelica* para compilar e executar os modelos internamente. Dessa forma, é possível que modelos criados nessas ferramentas apresentem incompatibilidade de versão com o Auto-FDI, uma vez que não usam o mesmo compilador.

Para a criação do ambiente de diagnóstico e a execução do diagnóstico (etapas 3 e 4) o Auto-FDI possui dois aplicativos gráficos chamados respectivamente de Auto-FDI *Editor* e Auto-FDI *Executor*.

A seguir descrevemos os passos necessários para a criação uma solução de diagnóstico completa desde o início. Para um melhor aproveitamento é importante que o leitor acompanhe o manual realizando as instruções no Auto-FDI.

O sistema que usaremos como exemplo é o sistema de excitação do tipo ST1A[1]. A função básica de um sistema de excitação é prover corrente contínua para enrolamentos de campos de máquinas síncronas. Além disso, os sistemas de excitação realizam funções de controle e proteção, através do controle da tensão e, conseqüentemente, da corrente da máquina. Tais funções são essenciais para o desempenho adequado dos sistemas de potência.

1.2 Iniciando o Auto-FDI

Ao iniciar o Auto-FDI será exibida a seguinte tela:

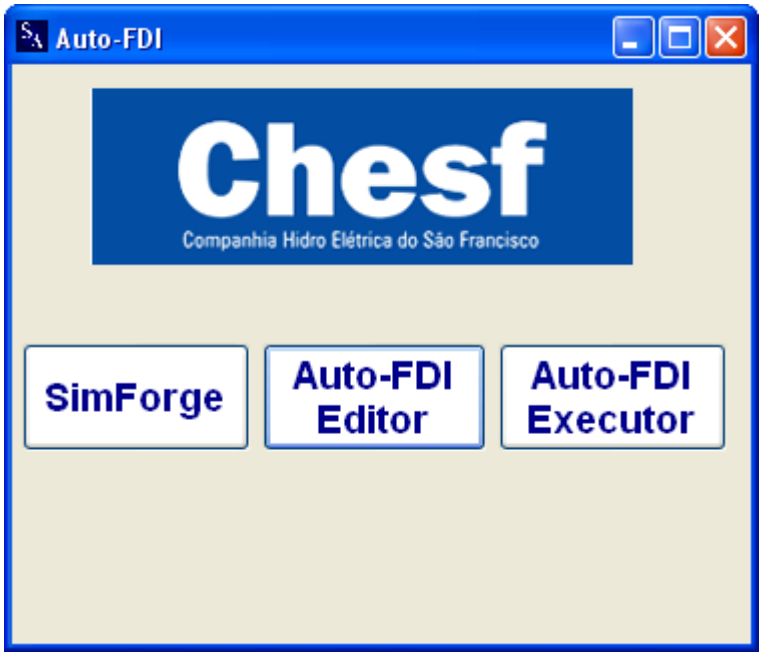


Figura 49: Tela inicial do Auto-FDI

Esta tela é o ponto de partida para a escolha entre os três ambientes de execução do *Auto-FDI*. A saber:

[1] Uma descrição esse sistema pode ser encontrada em [1]. O modelo para o sistema foi obtido de [2];

[1] Larsson, M., ObjectStab-an educational tool for power system stability studies, IEEE Transactions on Volume 19, Issue 1, Feb. 2004 Page(s): 56 - 63P

[2] Kundur, Power System Stability and Control, ser. Power System Engineering Series. McGraw-Hill, New York, 1994.

- ***SimForge*** – Editor para a criação e execução do modelo que descreve o sistema e do modelo de detecção de falhas;
- ***Auto-FDI Editor*** – Para criação do ambiente de diagnóstico;
- ***Auto-FDI Executor*** – Para execução do diagnóstico.

É importante ressaltar que esses ambientes são usados geralmente por diferentes usuários, de acordo com o perfil de cada usuário. Por exemplo, podemos ter um perfil de usuário "modelador" que usará o *SimForge* para a criação dos modelos e um outro perfil "operador" que irá realizar o diagnóstico através do *Auto-FDI Executor*.

1.3 Criação do Modelo

Nesta seção descrevemos os passos necessários para a criação do modelo do sistema de excitação. Conforme mencionamos anteriormente, o Auto-FDI vem com o *SimForge* integrado para a criação de modelos. Para informações detalhadas sobre o *SimForge*, consulte o manual da ferramenta através do menu "Help" da ferramenta ou no site da mesma.

Para abrir o *SimForge*, clique em

na tela inicial do *Auto-FDI*.

Após abrir o *SimForge*, clique na opção *File|New Project*. Escolha um nome para o projeto e selecione o caminho onde o projeto será criado, conforme mostra a figura abaixo.

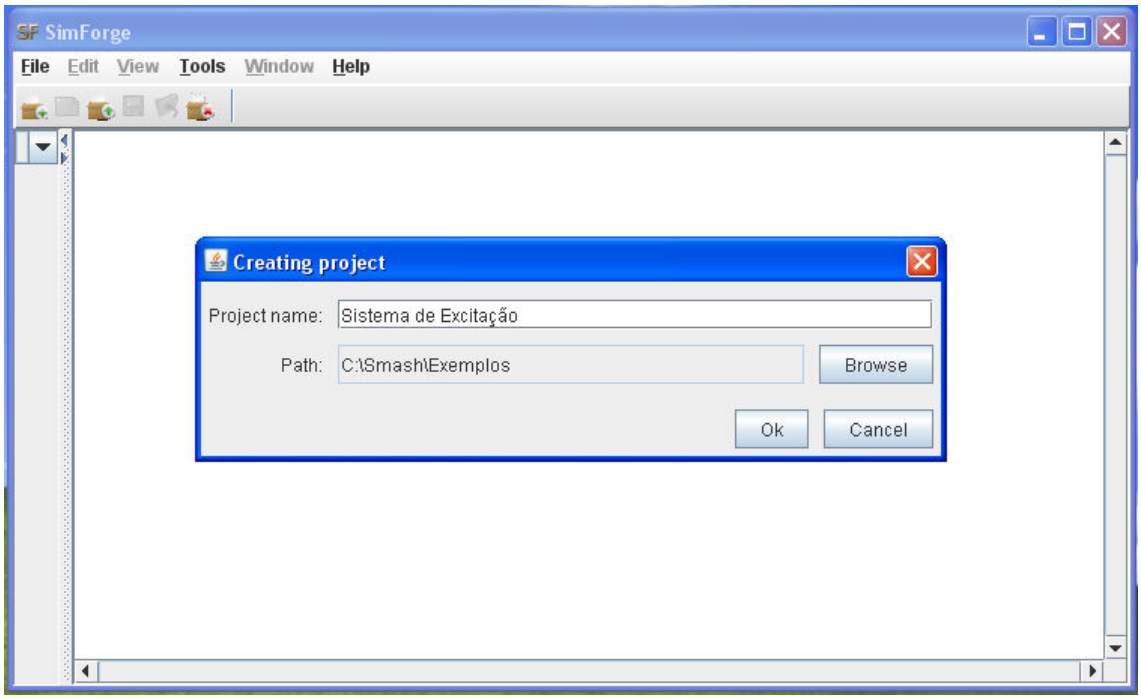


Figura 50: Criando um projeto novo no SimForge

O nome do projeto será a pasta onde todos os arquivos do projeto serão armazenados. Este deve ser um nome de diretório válido e **sem caracteres de espaço**.

Clique em "OK". Será criado o diretório do projeto, contendo:

- A pasta *IEC61131* – Usado para modelagem em linguagens IEC61131. Essa funcionalidade não é usada pelo *Auto-FDI*;
- A pasta *Modelica* – onde serão salvos os arquivos *Modelica* (.mo) do projeto;
- A pasta *Results* – para armazenamento dos resultados da simulação no *SimForge*;
- A pasta *Temp* – para arquivos temporários criados durante a simulação;
- O arquivo *Properties.xml*, que contém informações de configuração do projeto, tais como o diretório da biblioteca *Modelica* e do compilador *OpenModelica* que serão usados. Ao instalar o Auto-FDI, o *SimForge* já vem com uma configuração default. Essa configuração é usada para todos os projetos criados no *SimForge* (configuração geral) e pode ser mudada no menu *Tools|Settings*. Alternativamente a alteração da configuração para apenas um determinado projeto (configuração

específica) pode ser feita através do menu *Tools*|*ProjectProperties* depois que o projeto for criado.

Depois de criar o projeto, é exibida, no lado esquerdo do *SimForge,* a **janela de navegação de componentes** que permite acessar os componentes *Modelica* disponíveis no projeto e obter o resultado da simulação. Os componentes *Modelica* são organizados em pacotes. Para expandir os elementos da janela de navegação basta dar dois cliques no elemento. *Conforme já mencionamos anteriormente a opção IEC61131 não será usada e, portanto deve ser ignorada*.

Ao expandir o elemento "Modelica" na janela de navegação, encontramos os seguintes elementos:

- *Used external packages* - Onde estão disponíveis todos os pacotes que são usados pelo projeto, mas que não podem se editados, a exemplo da *Biblioteca Padrão Modelica* e outras bibliotecas de terceiros.

- *Modelica classes* – Onde se encontram os componentes *Modelica* criados no projeto e que podem ser editados.

- *Modelica Files* – Permite visualizar os arquivos Modelica (.mo) do projeto. Permite a edição apenas no modo textual.

Uma vez criado o projeto, vamos agora criar o modelo do sistema de excitação. Clique na opção do menu File|*new File* ou no ícone na barra de ferramentas do *SimForge*. Será exibida a tela ilustrada na Figura 49. Informe o nome e o tipo do arquivo. Escolha a opção "Modelica files", pois queremos criar um arquivo Modelica (.mo).

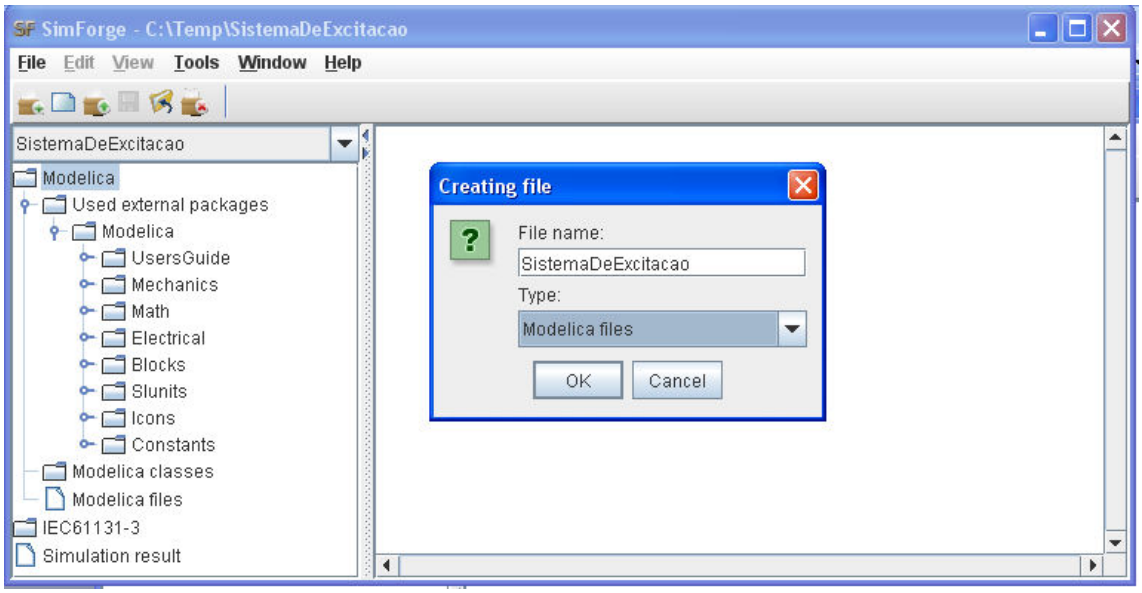


Figura 51: Criando um arquivo no *SimForge.*

Clique em “OK”. A próxima tela a aparecer permite escolher o tipo da classe a ser criada. A linguagem *Modelica* define vários tipos de classes, entre eles *model*, *package*, *connectors*, *block*, etc. Para informações sobre a semântica de cada um desses tipos consulte a documentação da linguagem.

Selecione o tipo “model” e em seguida clique em “OK”.

Para iniciar a construção do modelo é necessário dar dois cliques na classe criada (SistemaDeExcitacao), em “Modelica classes”, na janela de navegação de componentes. Feito isso irá aparecer duas janelas: uma permite a criação e edição do modelo graficamente e a outra no modo textual. A Figura 52 ilustra a área de montagem visual do modelo (área quadriculada). A janela para edição textual encontra-se por trás dessa janela.

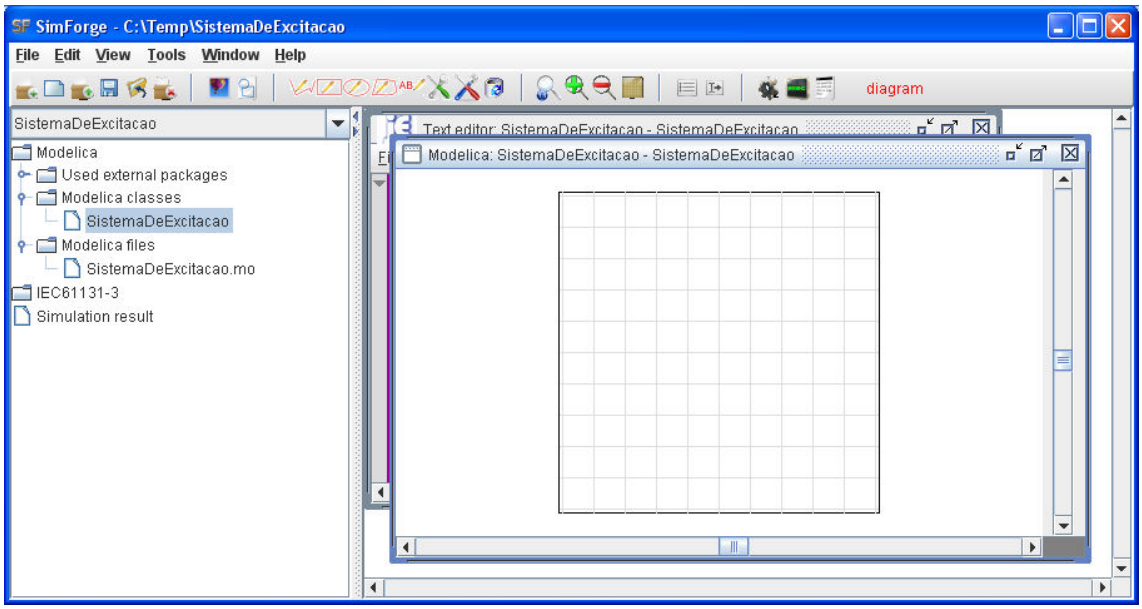


Figura 52: Área de montagem do modelo do *SimForge*.

Utilize as opções de zoom in/zoom out para aproximar e afastar a visualização do modelo. Também é possível ajustar o tamanho da área de montagem do modelo. Isto é muitas vezes desejável quando o modelo que queremos criar possui muitos componentes. A figura abaixo ilustra onde alterar o tamanho da área de montagem no modo de edição textual.

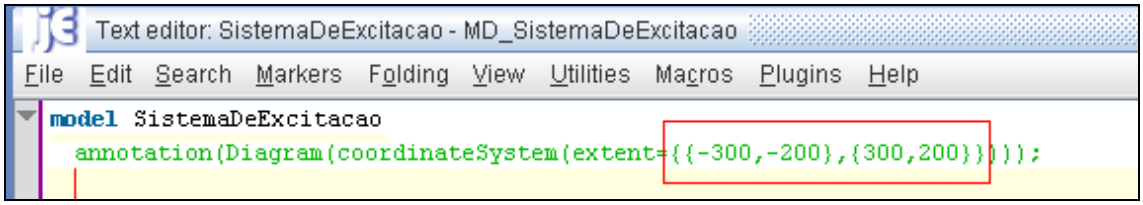


Figura 53: Ajustando o tamanho da área de montagem do modelo

Agora podemos iniciar a criação do modelo do sistema de excitação. Iremos criar o modelo visualmente, arrastando e interligando os componentes da biblioteca de componentes e funções *Modelica*. Para facilitar o acompanhamento das instruções, o resultado final da representação visual do modelo é mostrado na figura abaixo:

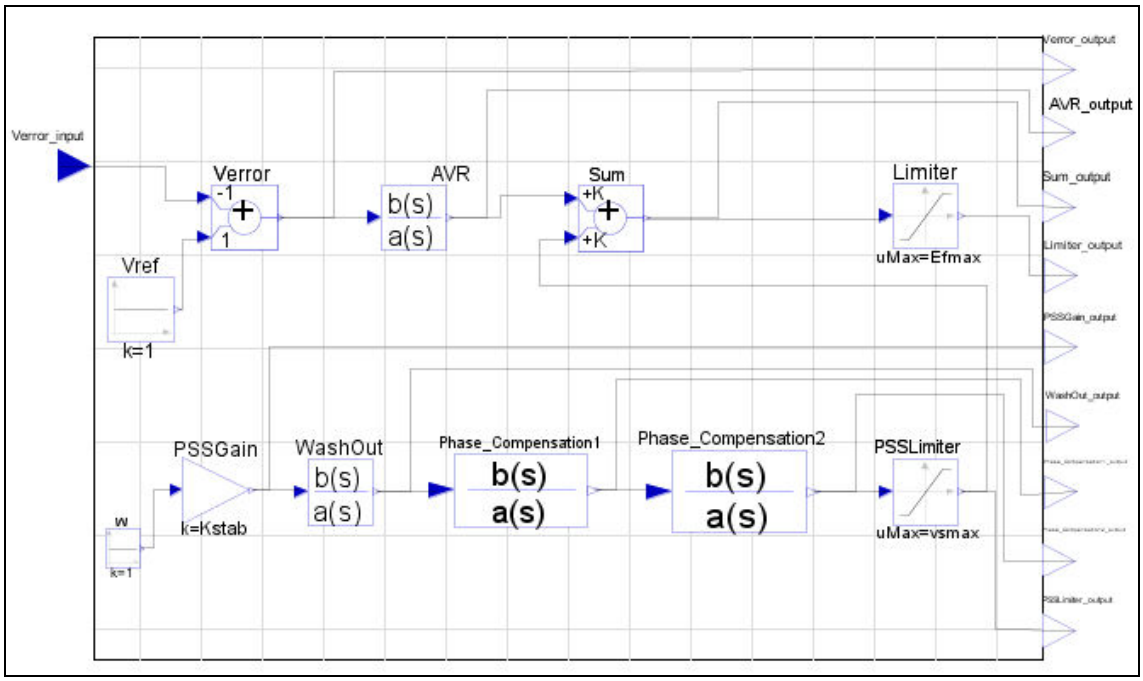


Figura 54: Representação visual do modelo do sistema de excitação do tipo ST1A

Primeiro selecione o componente do tipo "*Modelica.Blocks.Math.Gain*" e arraste para a área de montagem do modelo. Ao soltar o componente na área de montagem, deverá ser informado o nome do componente, conforme ilustra a figura abaixo:

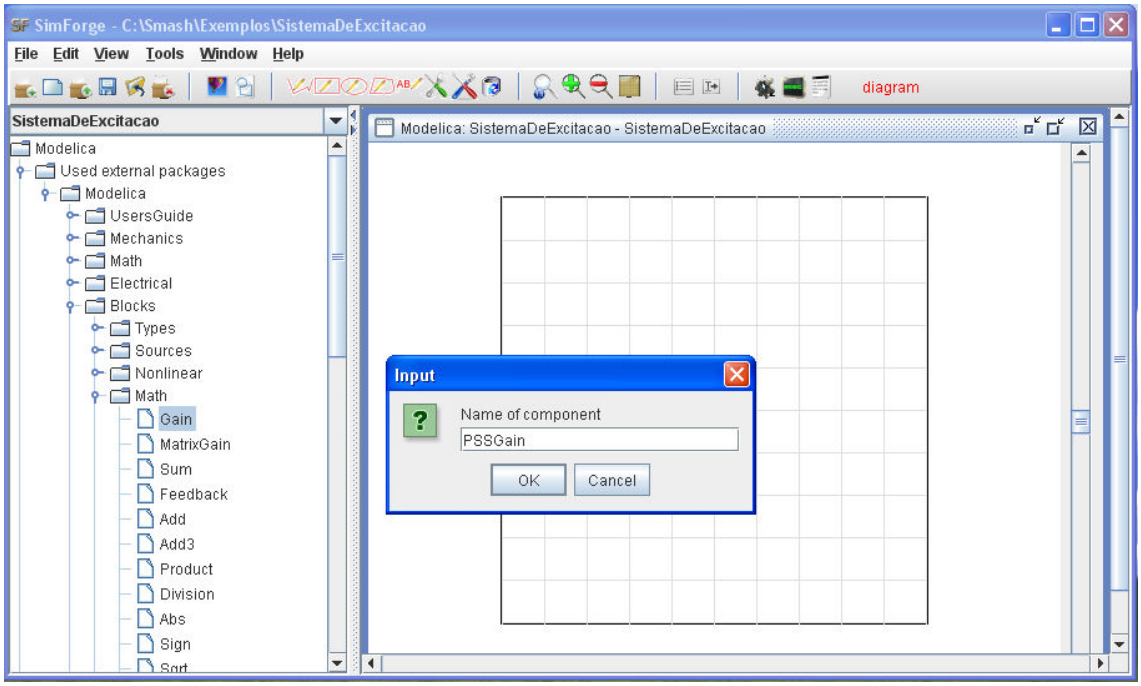


Figura 55: Inserindo o componente PSSGain

Clique em “OK”. O componente será exibido na área de montagem, conforme ilustra a Figura 56. É possível redimensionar um componente (ou vários) selecionando ele e arrastando suas bordas. Para selecionar vários componentes desenhe com o mouse um retângulo que inclua todos os componentes ou selecione-os com CTRL+clique do mouse em cada um dos componentes.

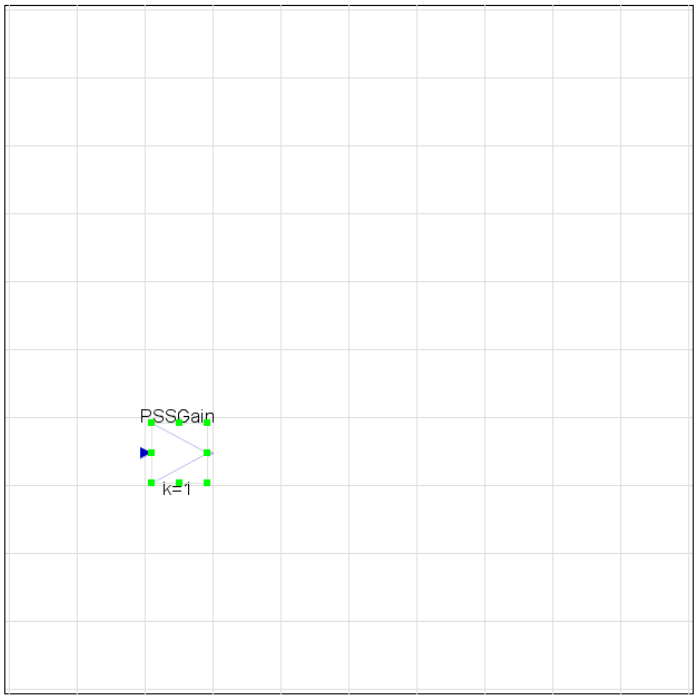


Figura 56: Componente *PSSGain* na área de montagem visual do modelo.

Da mesma maneira, arraste os outros componentes de modo a obter o modelo parcial equivalente ao ilustrado na Figura 57. O tipo de cada componente do modelo encontra-se na Tabela 5.

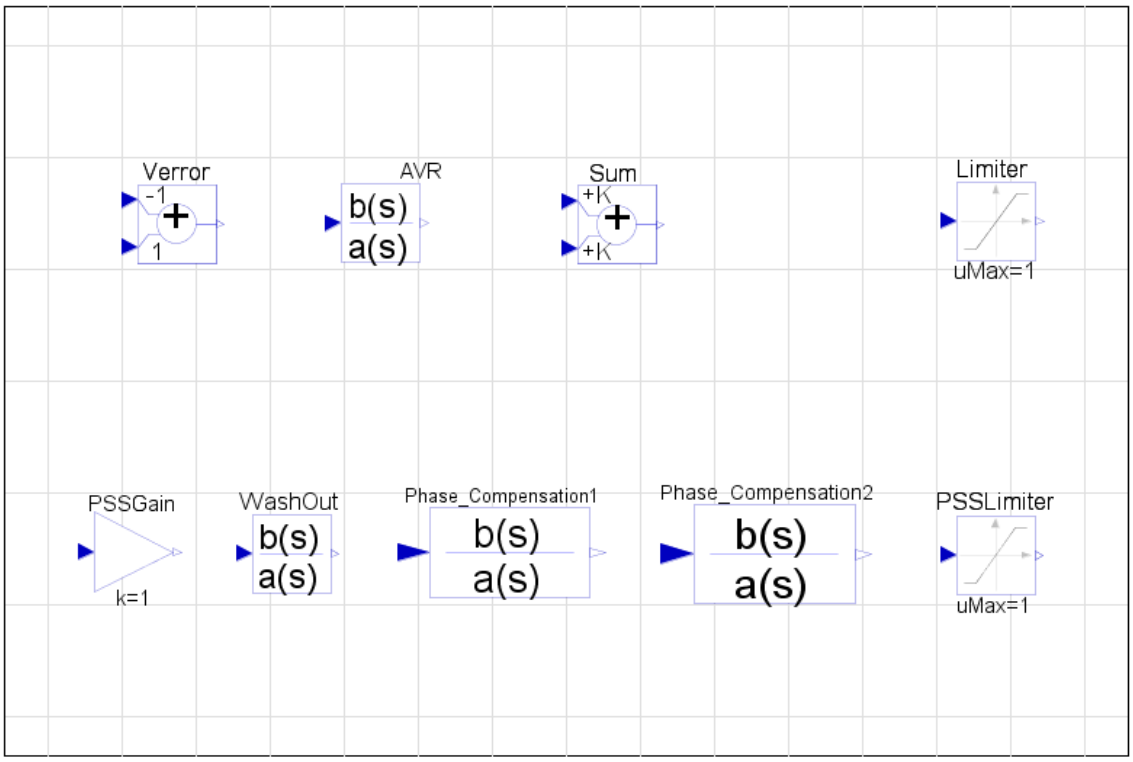


Figura 57: Modelo parcial do sistema de excitação.

Nome do Componente	Tipo do Componente
Verror	Modelica.Blocks.Math.Add
AVR	Modelica.Blocks.Continuous.TransferFunction
Sum	Modelica.Blocks.Math.Add
Limiter	Modelica.Blocks.Nonlinear.Limiter
PSSGain	Modelica.Blocks.Math.Gain
WashOut	Modelica.Blocks.Continuous.TransferFunction
Phase_Compensation1	Modelica.Blocks.Continuous.TransferFunction
Phase_Compensation2	Modelica.Blocks.Continuous.TransferFunction
PSSLimiter	Modelica.Blocks.Nonlinear.Limiter

Tabela 5: Tipo dos componentes do modelo do sistema de excitação

Agora vamos interligar os componentes. A

Figura 58 ilustra a conexão dos componentes *PSSGain* e *WashOut*. Primeiro posicione o cursos do mouse no conector do componente PSSGain e espere por um instante até que o nome do conector apareça (

Figura 58 – a). Em seguida, pressionando a tecla *Shift*, clique e solte o botão direito do mouse. Depois mova o cursor para o conector do componente *WashOut* e clique o mouse novamente (

Figura 58 – b). A representação para a conexão entre os componentes é uma linha entre eles. As propriedades visuais da linha (cor, espessura, etc.) podem ser alteradas através do botão direito do mouse na opção "Properties".

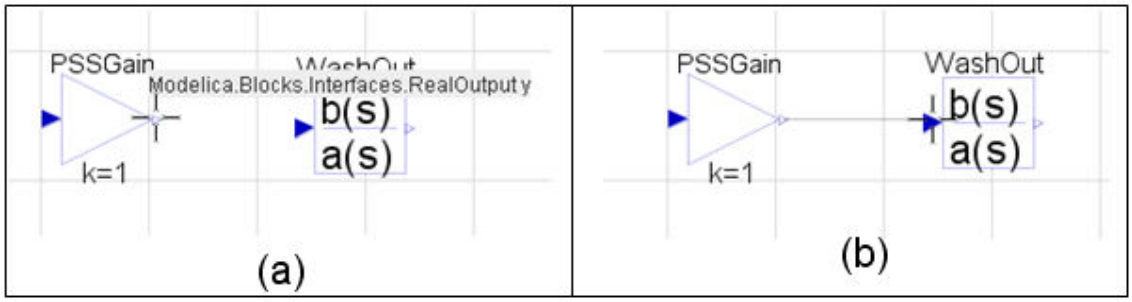


Figura 58: Conectando componentes no SimForge.

Conecte todos dos componentes de modo a obter o modelo ilustrado na abaixo:

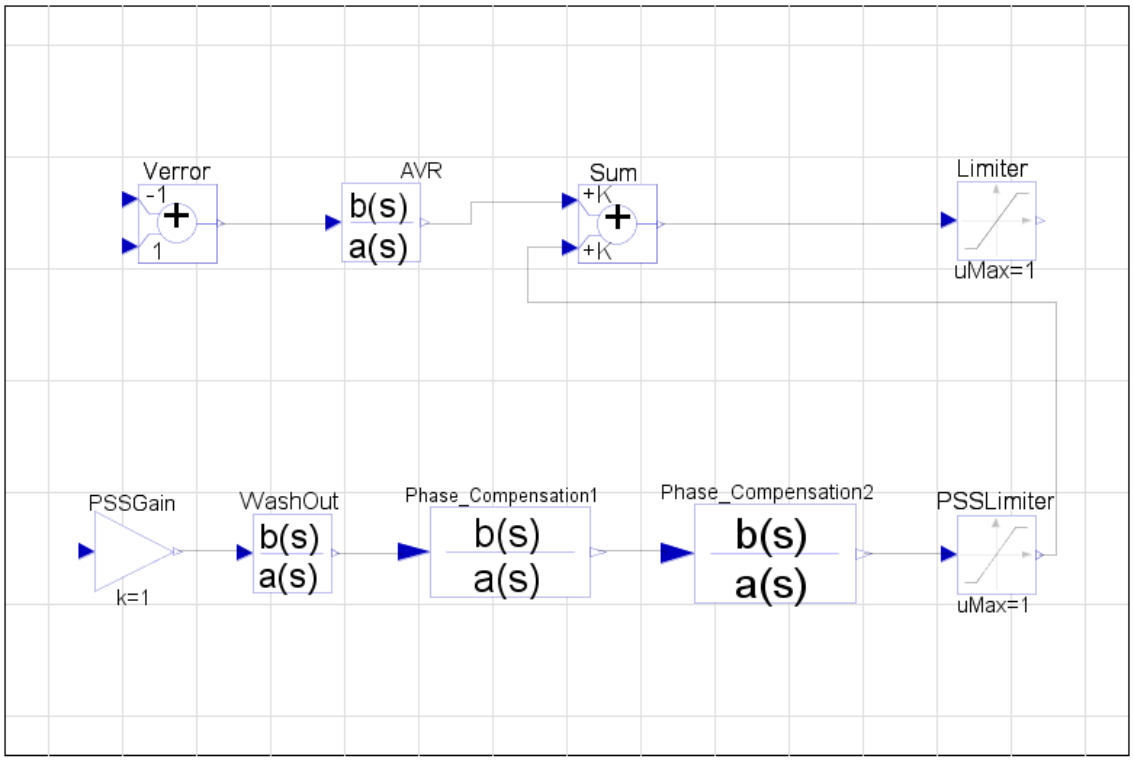


Figura 59: Modelo parcial do sistema de excitação

Vamos agora definir os valores default dos **parâmetros** dos componentes do nosso modelo, conforme a tabela abaixo[1]. Os parâmetros são valores que se mantém constante durante a simulação, mas que podem ser alterados entre uma simulação e outra. Conforme veremos mais adiante, também é possível alterar os valores dos parâmetros do modelo no *Auto-FDI Editor* e no *Auto-FDI Simulator*.

Parâmetro	**Valor**	**Descrição**
K	200	Steady-State Gain
Efmax	7	Maximum Field Voltage [p.u]
Efmin	0	Minimum Field Voltage [p.u]
Tr	1.5	AVR Time Constant
Kstab	20	Power System Stabilizer (PSS) Gain
Tw	10	Signal Washout Filter Time Constant

[1] Os valores dos parâmetros do modelo foram obtidos de [1].

[1] Larsson, M., ObjectStab-an educational tool for power system stability studies, IEEE Transactions on Volume 19, Issue 1, Feb. 2004 Page(s): 56 - 63P

T1	0.05	PSS Phase Compensation Time Constant 1
T2	0.02	PSS Phase Compensation Time Constant 2
T3	3	PSS Phase Compensation Time Constant 3
T4	5.4	PSS Phase Compensation Time Constant 4
vsmax	0.2	PSS Maximum Output Limit
vsmin	-0.2	PSS Maximum Output Limit

Tabela 6: Parâmetros do modelo do sistema de excitção

Ao invés passar os valores dos parâmetros diretamente na instância de cada componente, vamos colocar a definição de todos os parâmetros na classe *SistemaDeExcitacao* e passar os valores desses parâmetros para os componentes através das variáveis. A definição de todos os parâmetros do modelo encontra-se no trecho de código *Modelica* abaixo. Para inserir-lo o no modelo, mude a janela de montagem do modelo para o modo de edição textual (janela que fica por trás da edição gráfica) e cole o código conforme ilustra a Figura 60.

```
parameter Real K=200 "Steady-State Gain";
parameter Real Efmax=7 "Maximum Field Voltage [p.u]";
parameter Real Efmin=0 "Minimum Field Voltage [p.u]";
parameter Real Tr=1.5 "AVR Time Constant";
parameter Real Kstab=20 "Power System Stabilizer (PSS) Gain";
parameter Real Tw=10 "Signal Washout Filter Time Constant";
parameter Real T1=0.05 "PSS Phase Compensation Time Constant 1";
parameter Real T2=0.02 "PSS Phase Compensation Time Constant 2";
parameter Real T3=3 "PSS Phase Compensation Time Constant 3";
parameter Real T4=5.4 "PSS Phase Compensation Time Constant 4";
parameter Real vsmax=0.2 "PSS Maximum Output Limit";
parameter Real vsmin=-0.2 "PSS Maximum Output Limit";
```

Código 6: Trecho de código Modelica com definição dos valores dos parâmetros do modelo.

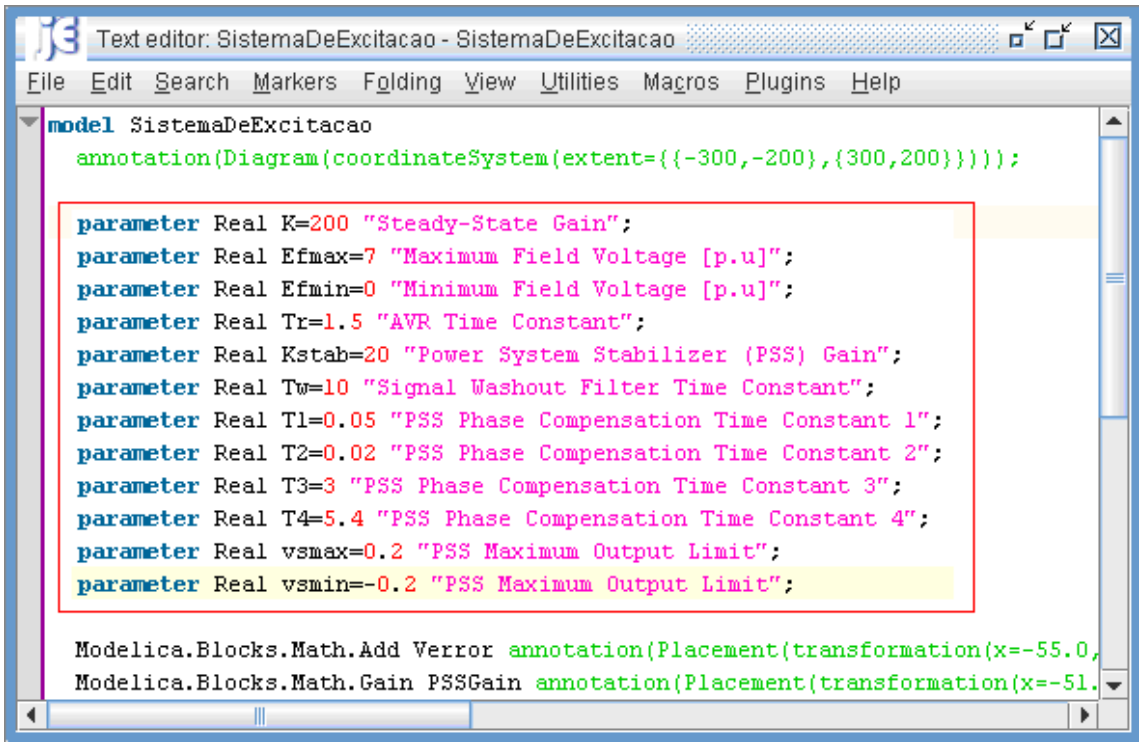


Figura 60: Código Modelica com a definição dos parâmetros.

Depois de definir o valor para os parâmetros, vamos agora passá-los para os componentes. Isto pode ser feito da seguinte maneira. Selecione o componente e clique com o botão direito do mouse sobre ele. Escolha a opção “Properties” e depois a aba “Parameters”. Coloque os valores dos parâmetros de acordo com a tabela abaixo. Quando um parâmetro é do tipo *array*, os elementos são colocados dentro de chaves, separados por vírgula (como por exemplo, o parâmetro “a” do componente “AVR” que recebe o valor {Tr, 1}) . A Figura 61 ilustra a alteração do valor do parâmetro do componente *Sum*.

Componente	**Parâmetros**
Verror	k1: -1 k2: 1
AVR	b: {1} a: {Tr,1}
Sum	k1: +K k2: +K
Limiter	uMax: Efmax

PSSGain	uMin: Efmin
WashOut	b: {Tw,0} a: {Tw,1}
Phase_Compensation1	b: {T1,1} a: {T2,1}
Phase_Compensation2	b: {T3,1} a: {T4,1}
PSSLimiter	uMax: vsmax uMin: vsmin

Tabela 7: Parâmetros dos componentes

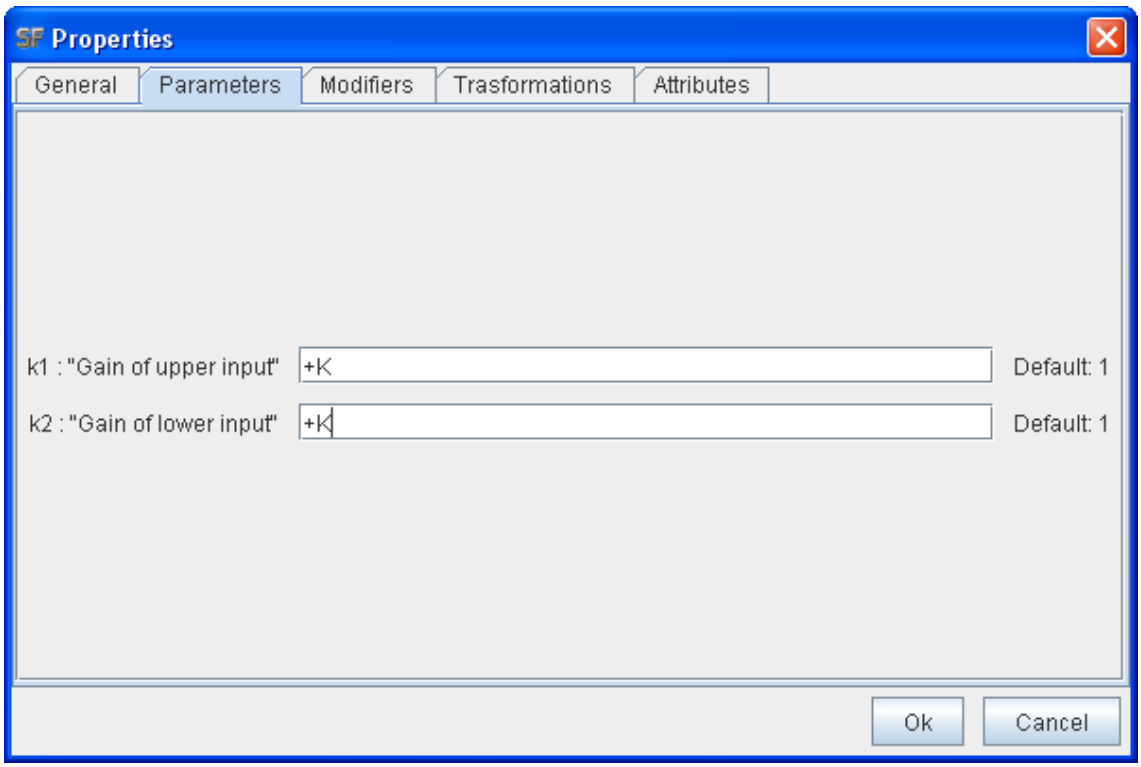


Figura 61: Alterando o parâmetro do componente *Sum*.

Feito isto, iremos definir agora a **interface de comunicação** do nosso modelo, ou seja, o meio pelo qual o modelo poderá se comunicar com o mundo externo. A comunicação entre componentes *Modelica* é feita através de **conectores**. As conexões só podem ser feitas entre conectores de tipos equivalentes. Para mais detalhes sobre o mecanismo de conexão *Modelica*, consulte a documentação da linguagem.

Somente as variáveis definidas na interface podem ser visíveis no *Auto-FDI* Editor e *Auto-FDI Executor*, conforme veremos mais adiante. Usaremos os componentes do tipo "*RealInput*" e "*RealOutput*" para definir a interface do modelo. Ambos componentes podem se encontrados no pacote "*Modelica.Blocks.Interfaces*" da biblioteca de componentes *Modelica*.

Iremos exteriorizar a saída de todos os componentes do modelo. Assim, arraste 9 componentes do tipo "RealOutput" e um do tipo "RealInput". Conecte as saídas de cada componente para o conector associado a ele, conforme ilustra a abaixo.

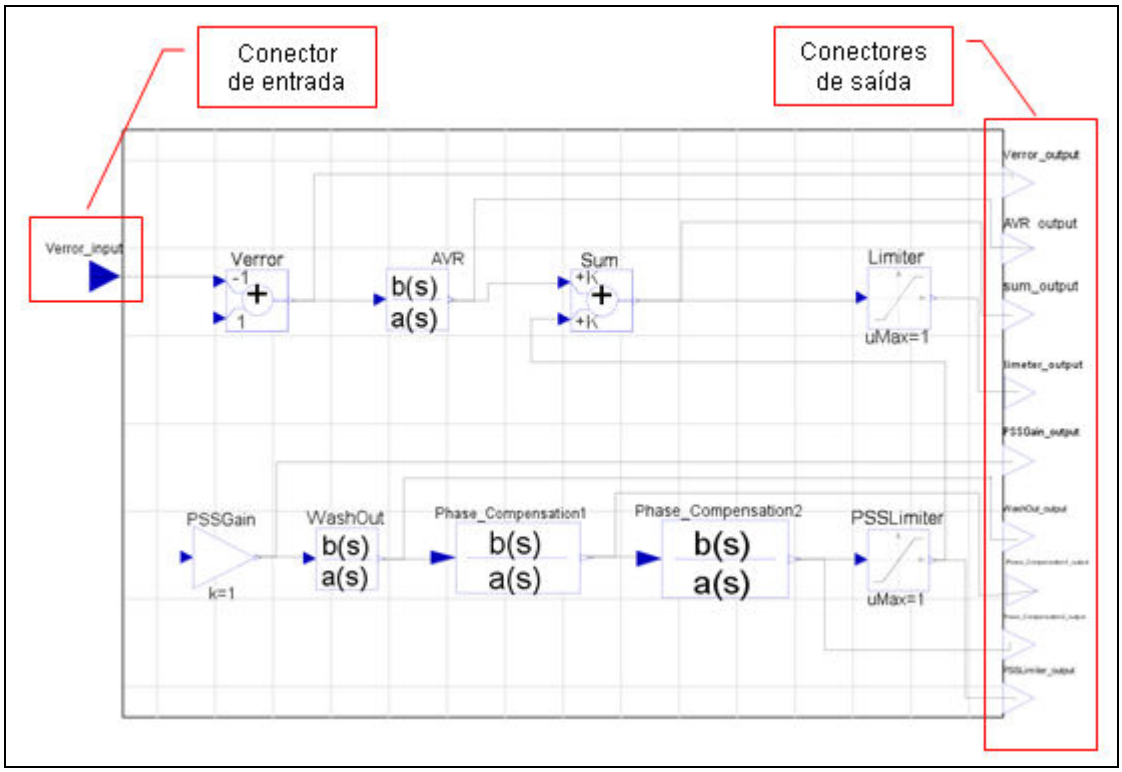


Figura 62: Modelo parcial do sistema de excitação

Agora defina a causalidade de cada conector colocado no modelo ("*input"* para o conector "Verror_input e "*output*" para os outros). Para isso selecione o conector, clique com o botão direito do mouse e escolha a opção "Properties" e depois a aba "Attributes". A Figura 63 ilustra o procedimento para o conector de entrada "Verror_input". Feito isso clique em "OK".

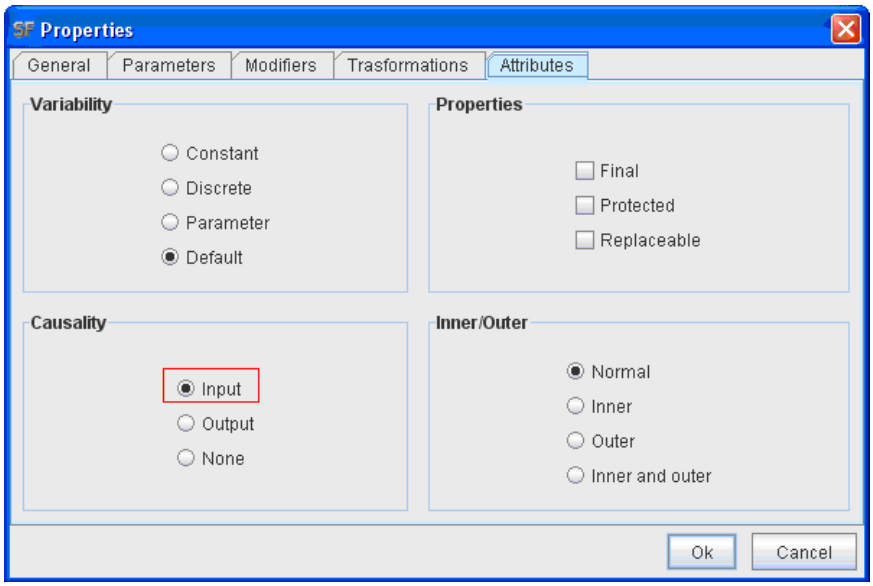


Figura 63: Definindo a causalidade do conector "Verror_input".

Observe na Figura 62 que o componente "Verror" possui duas entradas mais somente uma delas está conectada. Vamos usar um componente do tipo constante (Modelica.Blocks.Sources.Constant) para fazer a conexão com a outra entrada do componente "Verror". Também usaremos uma constante para conectar na entrada do componente PSSGain. Veja na figura abaixo.

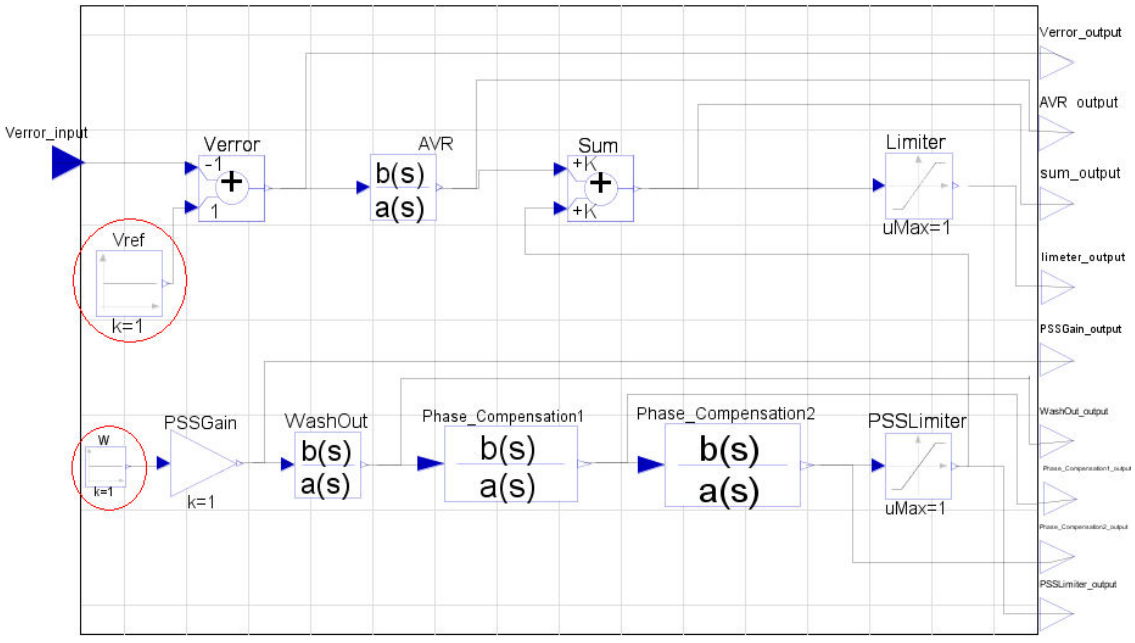


Figura 64: Modelo parcial do sistema de excitação

Por fim, iremos definir agora o **ícone** para o modelo criado. O ícone é a visualização do modelo por fora, ou seja, é como o modelo irá aparecer quando estiver sendo usado em outros modelos.

Clique no menu View/Icon ou no ícone na barra de ferramentas. Utilize as ferramentas de desenho no menu "Edit" ou na barra de ferramentas para desenhar o ícone para o modelo.

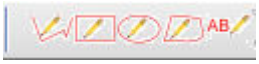


Figura 65: Ferramentas de desenho do SimForge

As ferramentas de desenho podem ser usadas não apenas para criar o ícone do modelo, mas também para desenhar no próprio modelo no modo de visualização gráfico. Um exemplo de ícone para o modelo é mostrado na Figura 66. Este ícone foi criado usando as ferramentas de desenho "Rectangle" e "Text". Para alterar as propriedades do desenho, selecione o desenho e escolha a opção "Properties" através do botão direito do mouse. Para mais detalhes consulte o manual do *SimForge*.

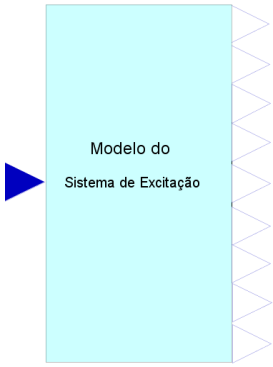


Figura 66: Ícone do modelo do sistema de excitação

Pronto! Terminamos a criação do Modelo. Para verifica se o modelo criado contém erros, Clique na opção *Tools*|*Check Model* do menu ou no ícone na barra de ferramentas. Se não houver erros no modelo, irá aparecer a mensagem abaixo e o modelo está pronto para ser simulado.

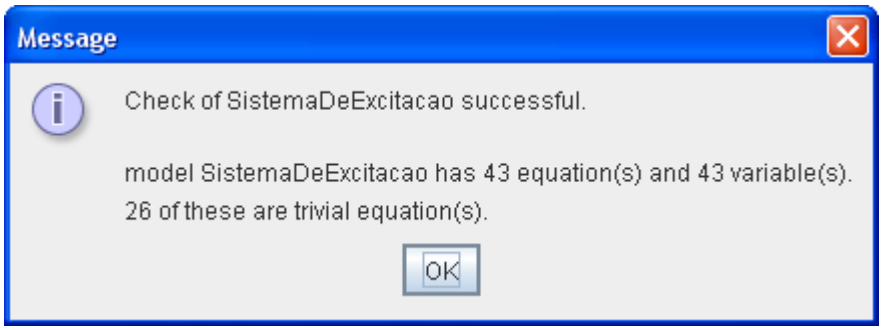


Figura 67: Mensagem de verificação de erros no modelo – Sem erros

Para simular o modelo no *SimForge*, clique na opção *Tools*|*Simulate* do menu ou no ícone na barra de ferramenta. Informe os valores 0 e 10 para o tempos inicial e final respectivamente conforme ilustra a figura abaixo.

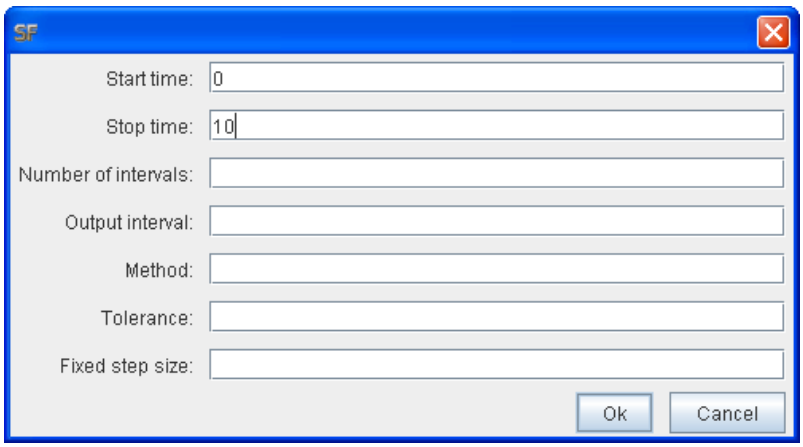


Figura 68: Opções para simulação do modelo

Clique em "OK". Os gráficos com o resultado das simulações podem ser visualizado clicando em "Simulation Result" na janela do lado esquerdo do *SimForge*. É possível escolher, entre qualquer variável do modelo, quais serão plotadas no gráfico. A Figura 69 mostra o resultado da simulação para o modelo do sistema de excitação.

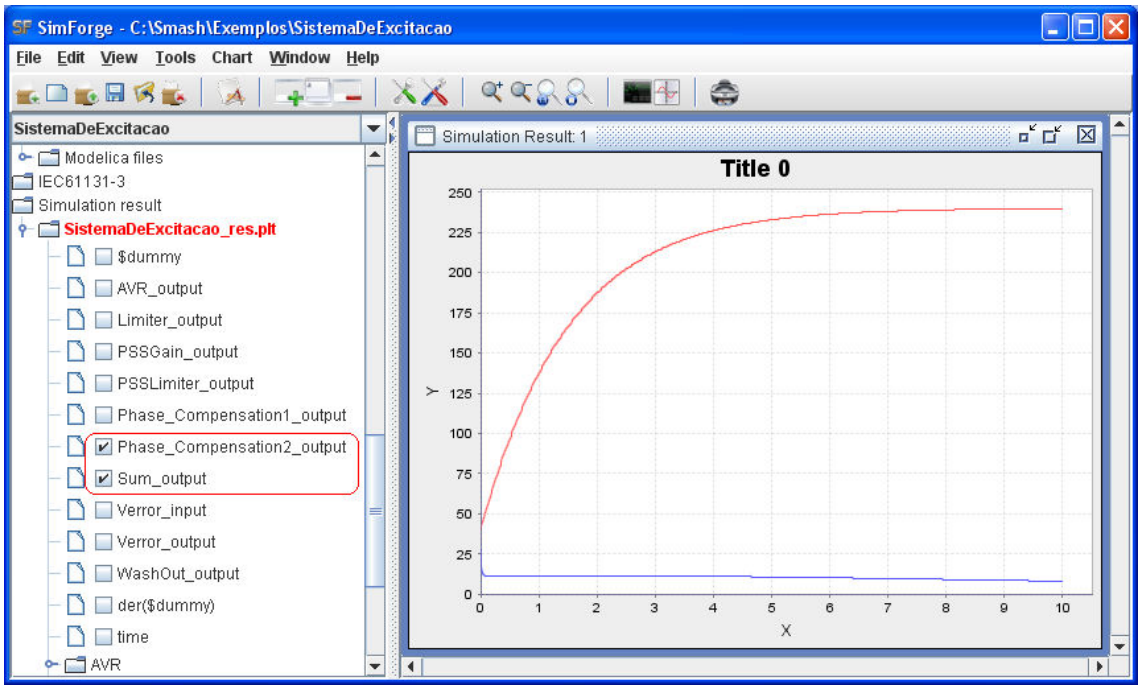


Figura 69: Verificação do resultado da simulação do modelo

1.4 Criação do Modelo de Detecção de Falhas

Depois de criado o modelo, o próximo passo é a criação do modelo que implementa a fase de detecção de falhas do processo de diagnóstico. A criação desse modelo, chamado de **Modelo de Detecção,** inclui a leitura dos sinais de entrada e saída do sistema, a geração de resíduos e a avaliação de resíduos. Detalhes sobre o processo de detecção e diagnóstico de falhas pode ser encontrado capítulo 2 desta dissertação.

A **injeção de sinais** do sistema no modelo é feita através de componentes especiais, chamados **adaptadores**. Os adaptadores são componentes *Modelica* que sabem como acessar os dados a serem consumidos pelo modelo. Alguns componentes adaptadores estão disponibilizados na biblioteca de componentes do *Auto-FDI*, a exemplo do *LeitorDeArquivos*. Para mais informações sobre a injeção de sinais ou sobre a criação de adaptadores, consulte o manual de referência do *Auto-FDI*. Também estão disponíveis na biblioteca de componentes do *Auto-FDI,* componentes especiais para a geração e avaliação de resíduos.

Vamos usar o mesmo projeto do *SimForge*, criado anteriormente, para a criação do modelo de detecção de falhas. O nome do modelo que iremos criar será

"MD_SistemaDeExcitacao". Do mesmo modo que já foi feito antes para criar um novo modelo, clique no ícone na barra de ferramentas do *SimForge* e informe o nome (MD_SistemaDeExcitacao) e o tipo do arquivo (modelica files). Clique em "OK" e em seguida escolha "model" para o tipo da classe a ser criada e clique em "OK". Para abrir o modelo e iniciar a construção, dê dois cliques na classe criada que encontra-se em "Modelica classes" na janela de navegação de componentes (lado esquerdo do *SimForge*). Feito isso será aberta a área de montagem do modelo.

Agora podemos iniciar a criação do modelo de detecção de falhas. O resultado final da representação visual do modelo é mostrado na Figura 70. Conforme podemos observar na figura, o modelo de detecção de falhas contém o modelo do sistema, os adaptadores para injeção de sinais do sistema e os componentes de geração e de avaliação de resíduos.

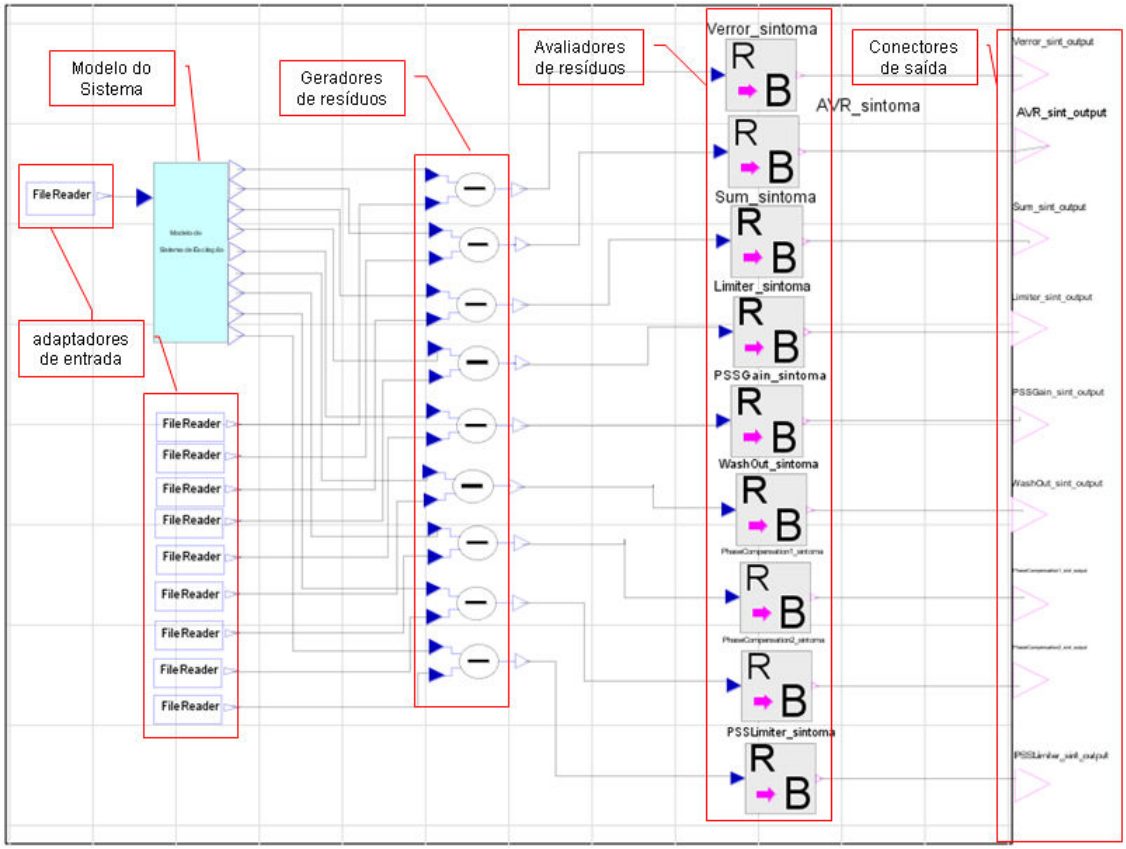


Figura 70: Representação visual do modelo de detecção de falhas

Primeiro vamos arrastar o modelo do sistema (criado anteriormente). Caso não lembre, o modelo do sistema está em "Modelica classes" na janela de navegação de

componentes. Informe “modelo” para o nome do componente e posicione/redimensione o componente conforme desejado. A Figura 71 mostra a área de montagem após a inclusão do modelo do sistema. Observe que aparece o ícone que criamos anteriormente para o modelo.

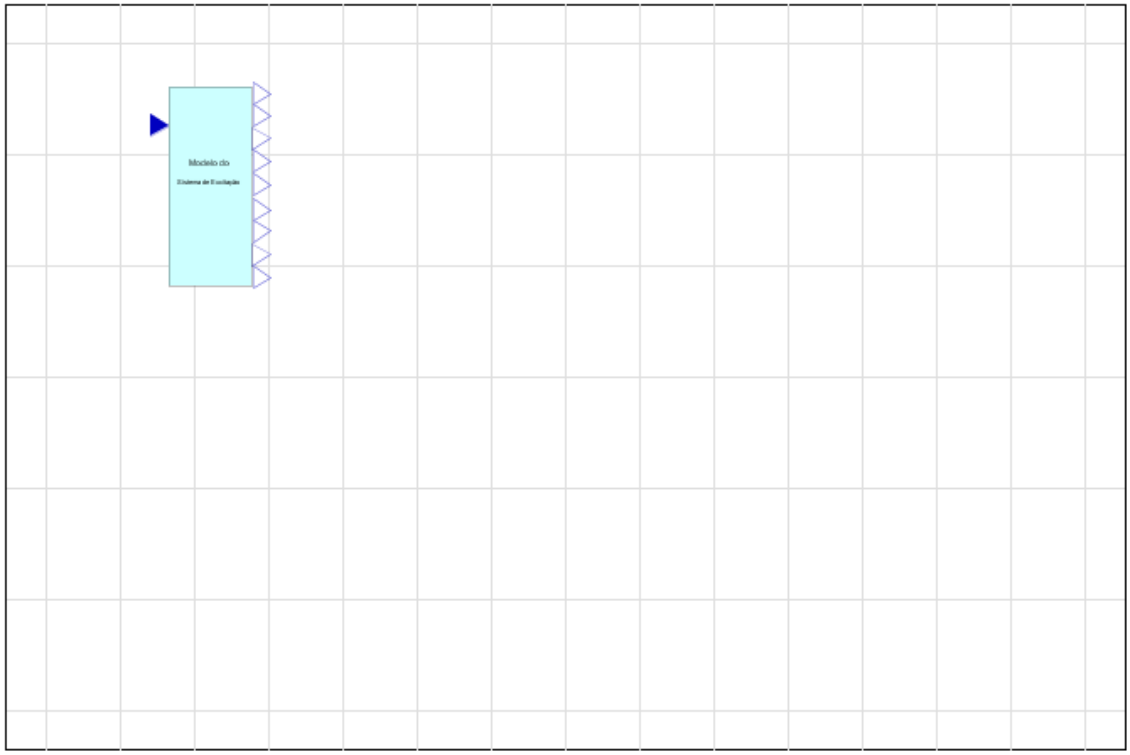


Figura 71: Modelo de detecção de falhas – adição do modelo do sistema

Agora vamos incluir no modelo de detecção os adaptadores de entrada. Conforme já dissermos anteriormente, os adaptadores permitem a injeção dos sinais obtidos do sistema. Estes sinais serão comparados com os valores obtidos do modelo do sistema a fim de que sejam gerados os resíduos.

O adaptador que usaremos será o LeitorDeArquivos que está disponível na biblioteca de componentes do *Auto-FDI* (Diagnostics). Um componente LeitorDeArquivo obtém os valores de apenas uma variável do sistema. Assim, arraste 10 componentes LeitorDeArquivo - um para cada variável de saída do sistema (“Verror_adapter”, “AVR_adapter”, “Sum_adapter”, “Limiter_adapter”, “PSSGain_adapter”, “WashOut_adapter”, “PhaseCompensation1_adapter”, “PhaseCompensation2_adapter”, “PSSLimiter_adapter”) e mais um para a variável de entrada (“Verror_in_adapter”, usado para alimentar o modelo). O resultado pode ser visto na figura abaixo:

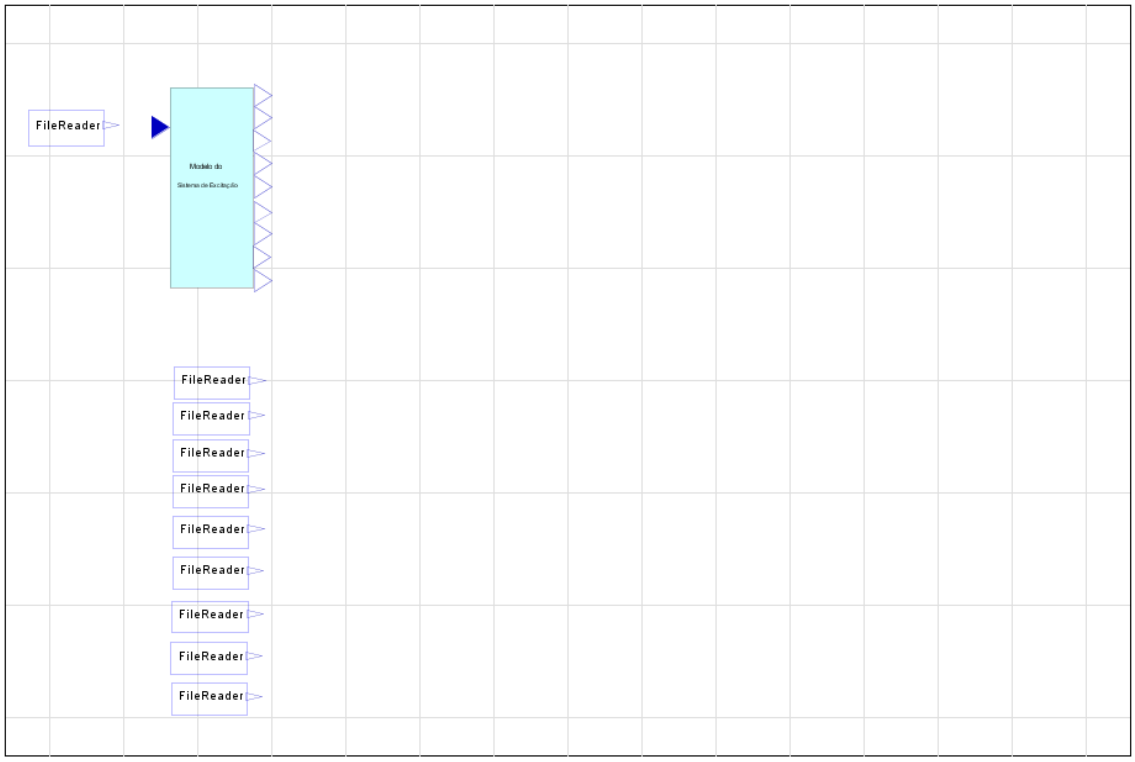


Figura 72: Modelo de detecção de falhas – adição dos adaptadores

O próximo passo é especificar o arquivo para cada adaptador de entrada. O caminho do arquivo é um parâmetro do componente. Componentes LeitorDeArquivo lê arquivos em um formato específico, com duas colunas. Cada linha do arquivo é dividida em duas colunas separadas por caractere de espaço. A primeira coluna contém valores numéricos, representando os tempos de medição. A segunda coluna contém os valores da variável medidos ao longo tempo. Os arquivos que usaremos no nosso exemplo estão disponíveis no diretório de instalação do *Auto-FDI*, na pasta "exemplos/sistema_de_excitacao/arquivos".

Primeiro copie a pasta de arquivos de exemplo para o projeto que criamos (C:\ *Auto-FDI*\Exemplos\SistemaDeExcitacao). Cada arquivo contém os valores de uma variável do sistema monitorado. Os arquivos são nomeados de acordo com os componentes do modelo (por exemplo, Verror.txt, AVR.txt, etc.).

Para especificar o arquivo de cada componente, selecione o componente e clique no botão direito do mouse. Escolha "Properties" e depois a aba "Parameters". No parâmetro "path" informe o caminho do arquivo, conforme é ilustrado na Figura 73 para

o componente “AVR_adapter”. Observe que, como o parâmetro é do tipo string, o valor deve está entre aspas duplas.

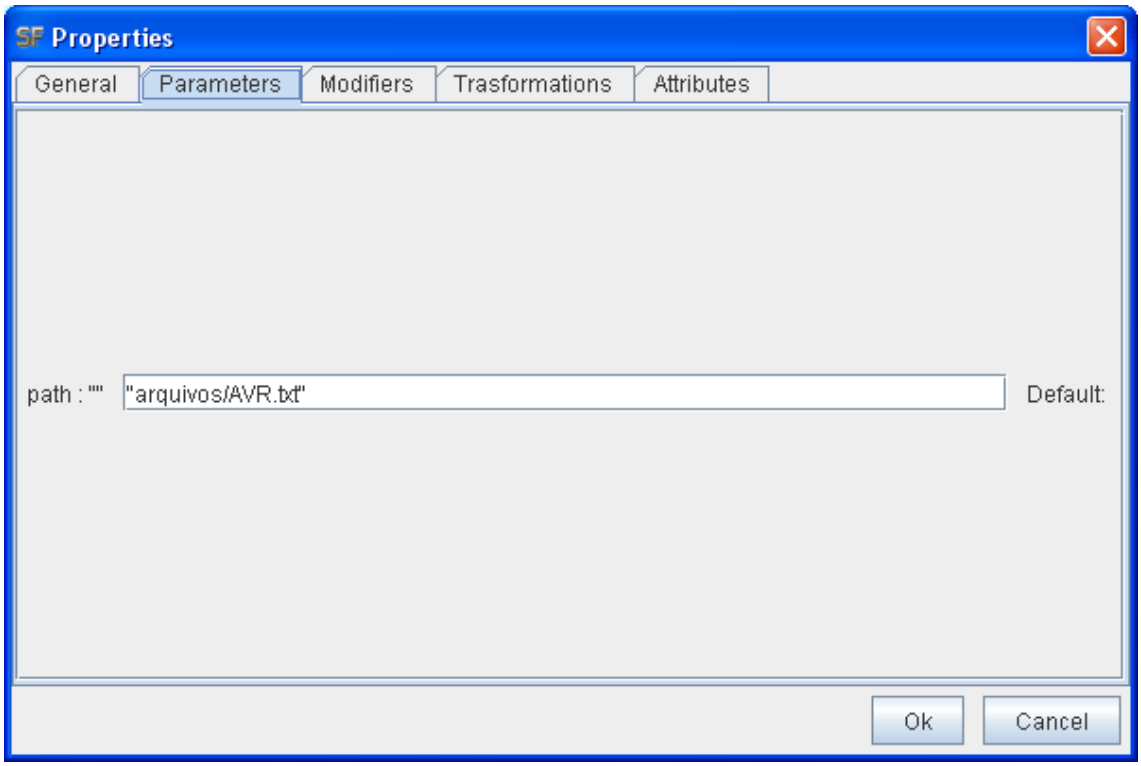


Figura 73: Definindo o arquivo de entrada para o adptador

Agora vamos implementar a geração e a avaliação de resíduos. A biblioteca de componentes do *Auto-FDI* disponibiliza alguns componentes para a geração e a avaliação de resíduos. Para a geração de resíduos usaremos o componente “Diagnostics.SubtractModel”. O SubtractModel gera o resíduo através de uma subtração entre o sinal obtido do sistema e o sinal obtido do modelo. Para a avaliação de resíduos usaremos um componente da biblioteca padrão da linguagem Modelica, o “Modelica.Blocks.Math.RealToBoolean”. O componente RealToBoolean compara o valor de entrada (que no caso será o resíduo) com um limiar, onde o limiar é um parâmetro do componente. A Figura 74 ilustra o modelo depois de colocados os componentes de geração e de avaliação de resíduos.

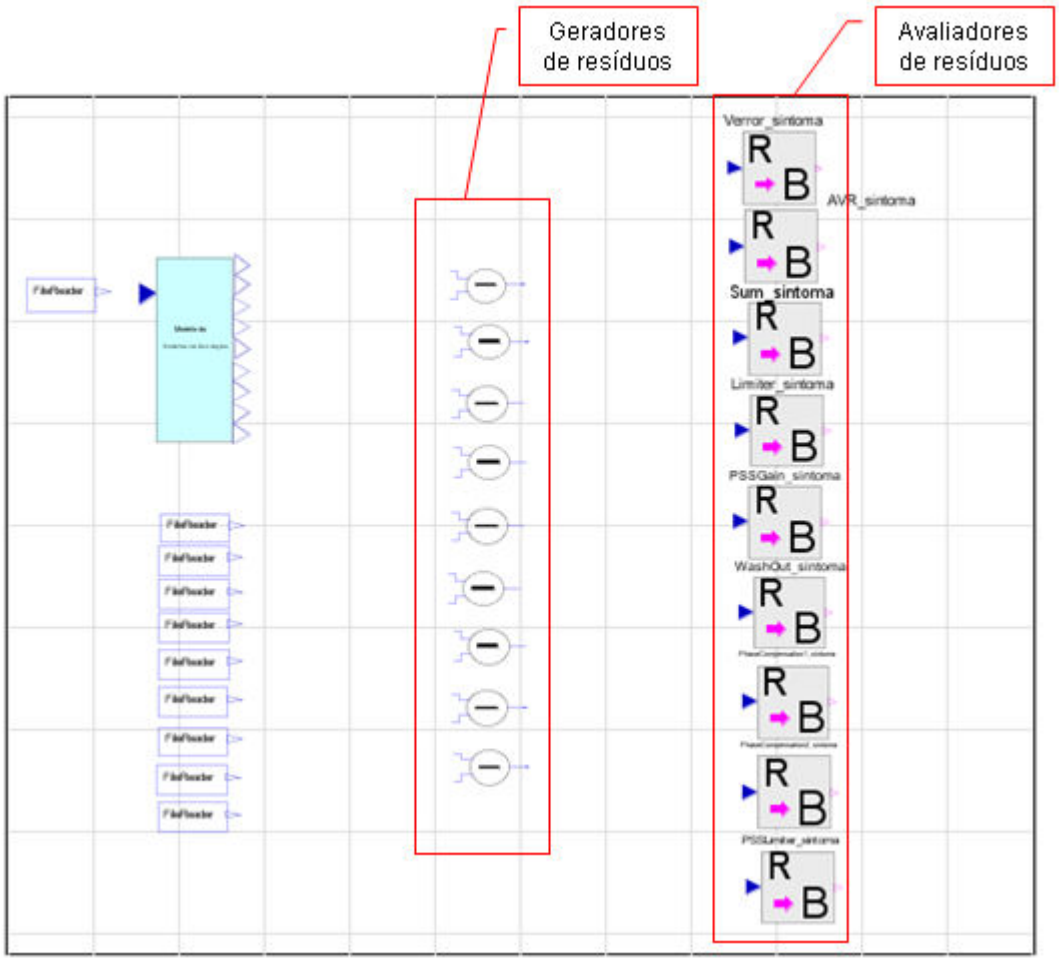


Figura 74: Modelo de detecção de falhas –
adição dos geradores e dos adaptadores de resíduos

Defina um limiar de "0.5" para os componentes de avaliação de resíduos.

Para exteriorizar as saídas dos componentes, usamos os conectores do tipo "Modelica.Blocks.Interfaces.BooleanOutput" disponíveis na biblioteca de componentes Modelica. Para alterar a causalidade de cada conector para "output, clique com o botão direito do mouse e escolha a opção "Propriedades" e depois a aba "Attributes", assim como foi feito para os conectores do modelo do sistema na seção anterior deste manual.

Conecte os componentes de modo a obter o modelo ilustrado na Figura 49.

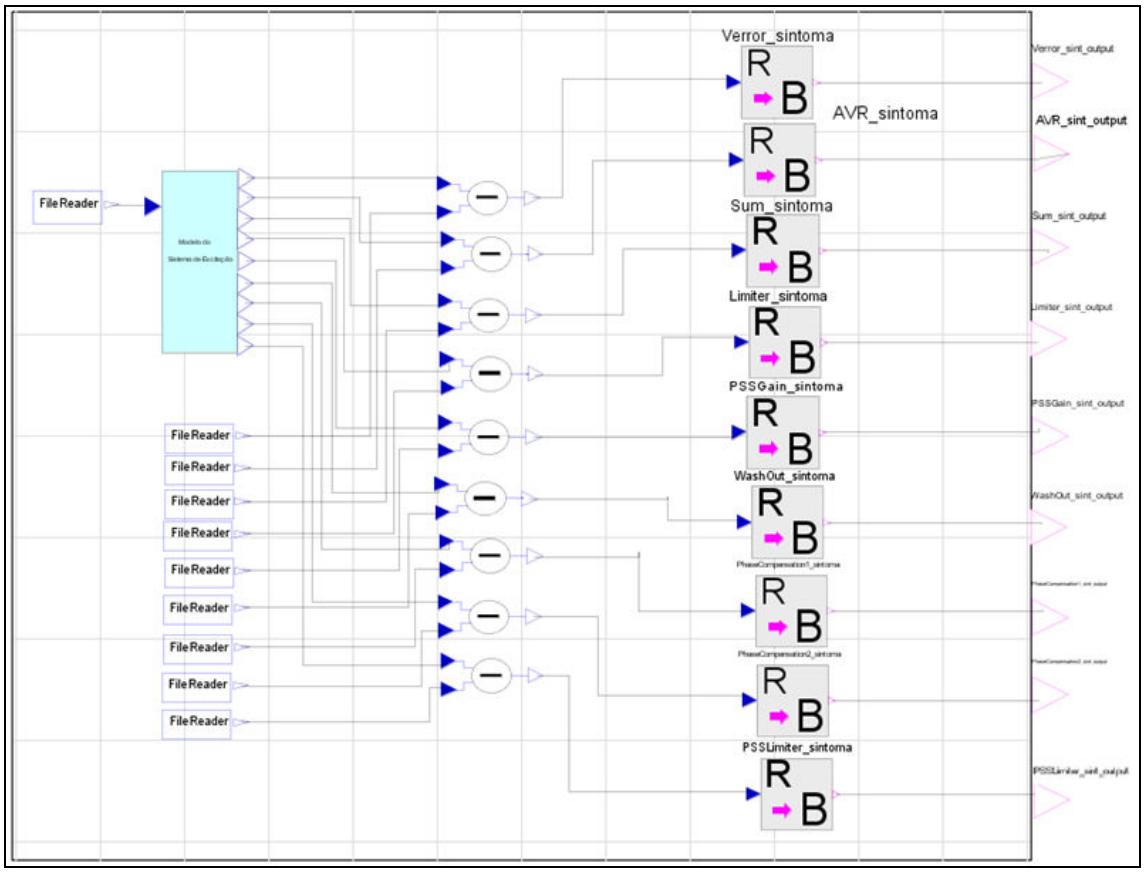


Figura 75: Modelo de detecção de falhas finalizado

Por fim, verifique se há erros no modelo, através da opção *Tools*|*Check Model* do menu ou no ícone na barra de ferramentas. Se não houver erros, será exibida a tela da Figura 76 e o modelo está pronto para ser usado na próxima etapa: a criação do ambiente de diagnóstico.

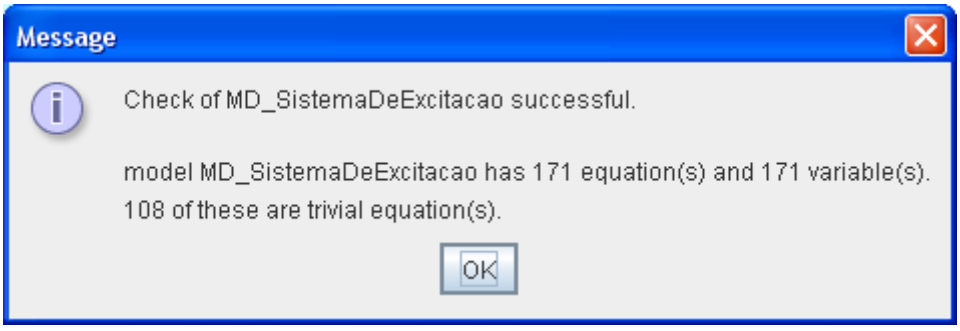


Figura 76: Mensagem de verificação de erros no modelo de detecção – Sem erros

1.5 Criação do ambiente de diagnóstico

Nesta seção iremos criar o ambiente de diagnóstico. O ambiente de diagnóstico é a interface gráfica que será criada para ser usada pelos operadores na realização do diagnóstico de um sistema. A criação do ambiente de diagnóstico é realizada no *Auto-FDI Editor*. O *Auto-FDI Editor* permite, entre outras funcionalidades:

- Criar o uma **interface altamente personalizada** de acordo com o sistema a ser avaliado. A interface pode incluir imagens do sistema e equipamentos, figuras, links, etc.;
- Adicionar gráficos para observar todo o processo de diagnóstico (valores entradas e saídas do sistema, resíduos, sintomas, falhas, etc.).
- Adicionar alarmes indicadores de falhas ou de sintomas de falha;
- Alteração dos parâmetros dos modelos (do sistema e de detecção de falha) e escolha de quais destes parâmetros poderão ser visíveis e/ou editáveis no ambiente de execução do diagnóstico;

Neste documento não iremos abordar todas as funcionalidades do *Auto-FDI Editor*. Para obter mais informações consulte o manual da ferramenta através do menu "help".

A figura abaixo mostra o ambiente de diagnóstico que iremos criar para o sistema de excitação.

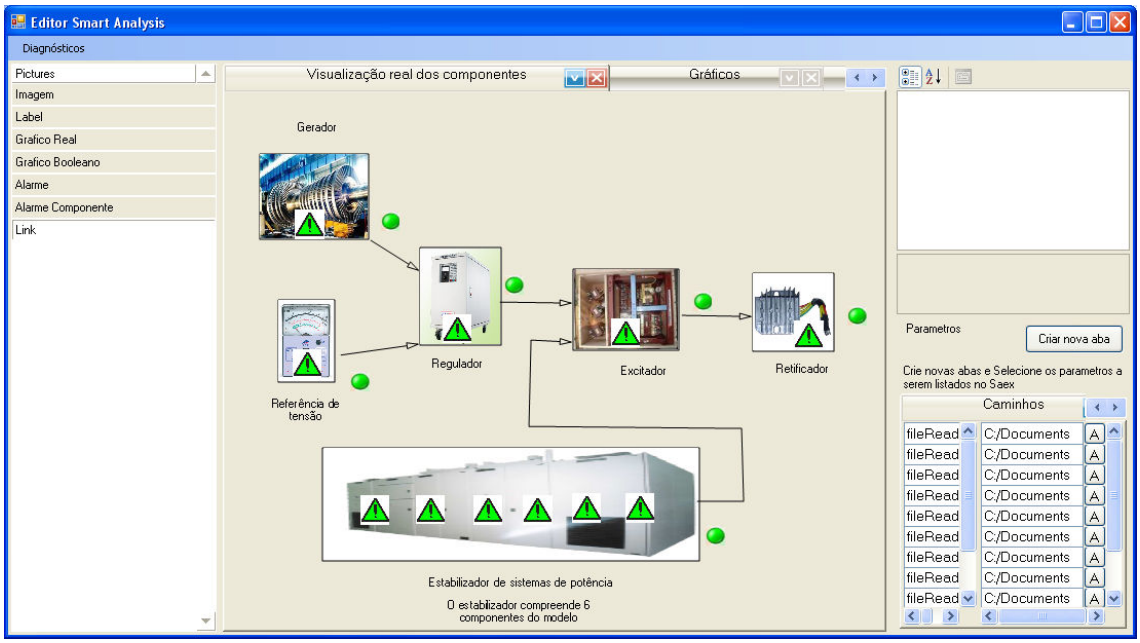


Figura 77: Ambiente de diagnóstico para o sistema de excitação

Antes de iniciar a construção do ambiente de diagnóstico é necessário copiar para o diretório de instalação do *Auto-FDI* os modelos que criamos anteriormente. O modelo do sistema (SistemaDeExcitacao.mo) junto com a pasta "arquivos" devem ser copiados para a pasta "Models" e o modelo de detecção (MD_SistemaDeExcitacao.mo) para a pasta "DetectionModels". Feito isto, os modelos poderão ser usados na criação do ambiente de diagnóstico e posteriormente, na execução do diagnóstico.

Para iniciar o *Auto-FDI Editor* clique na opção Smash Editor na tela inicial do *Auto-FDI*. O *Auto-FDI Editor* será aberto.

Depois de abrir o *Auto-FDI Editor*, escolha a opção *Diagnósticos|Novo Diagnosticador* no menu. Será a exibida uma tela para a escolha do Modelo e do Modelo de Detecção a serem usados. Selecione "SistemaDeExcitação" e "ModeloDeDetecao_SistemaDeExcitacao", que são respectivamente o modelo e o modelo de detecção criados nas seções anteriores deste manual. Veja a Figura 78. Perceba que ao selecionar o modelo a ser usado, só serão listados os Modelos de Detecção que utilizam o modelo selecionado.

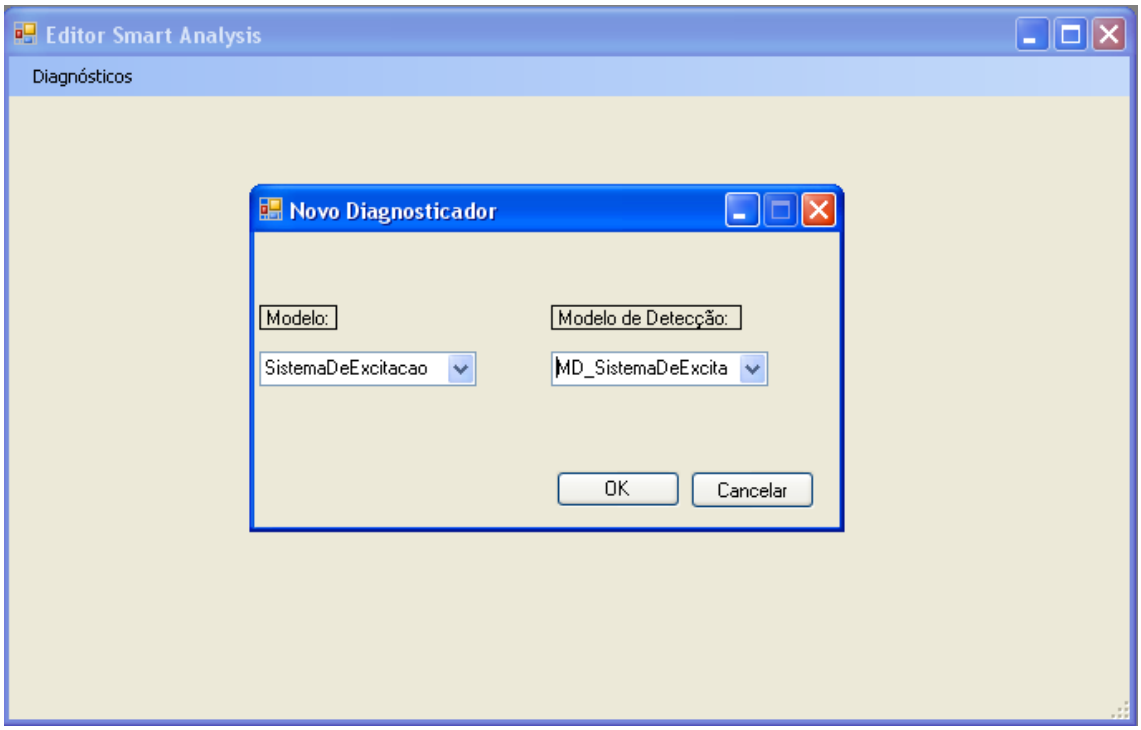


Figura 78: Escolhendo o modelo do sistema e o modelo de detecção no Auto-FDI Editor

Ao clicar em "OK" o editor será aberto com a área de representação visual do sistema vazia, conforme mostra a Figura 49Figura 79.

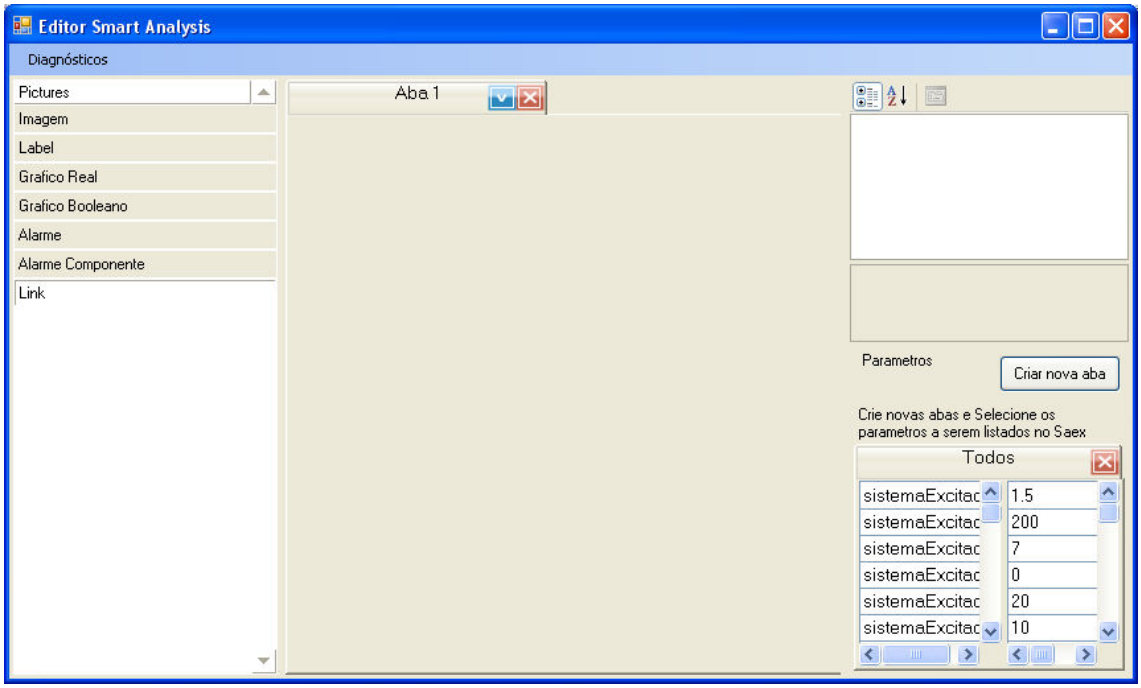


Figura 79: Auto-FDI Editor – Elementos da interface do editor.

Vamos iniciar criando a representação visual do sistema, como mostrado na Figura 77. Para colocar as imagens na área de desenho, arraste componentes do tipo "Imagem" da barra de ferramentas (lado esquerdo do editor). Para alterar as imagens default dos componentes, clique com o botão direito e escolha a opção "mudar imagem", conforme a mostra a Figura 80. As imagens do exemplo usado neste manual encontram-se no diretório de instalação do *Auto-FDI*, na pasta "exemplos/sistema_de_excitacao/imagens". Use os componentes do tipo "Link" e "Label" para ligar as imagens e colocar textos respectivamente de modo a obter a representação visual ilustrada na Figura 77.

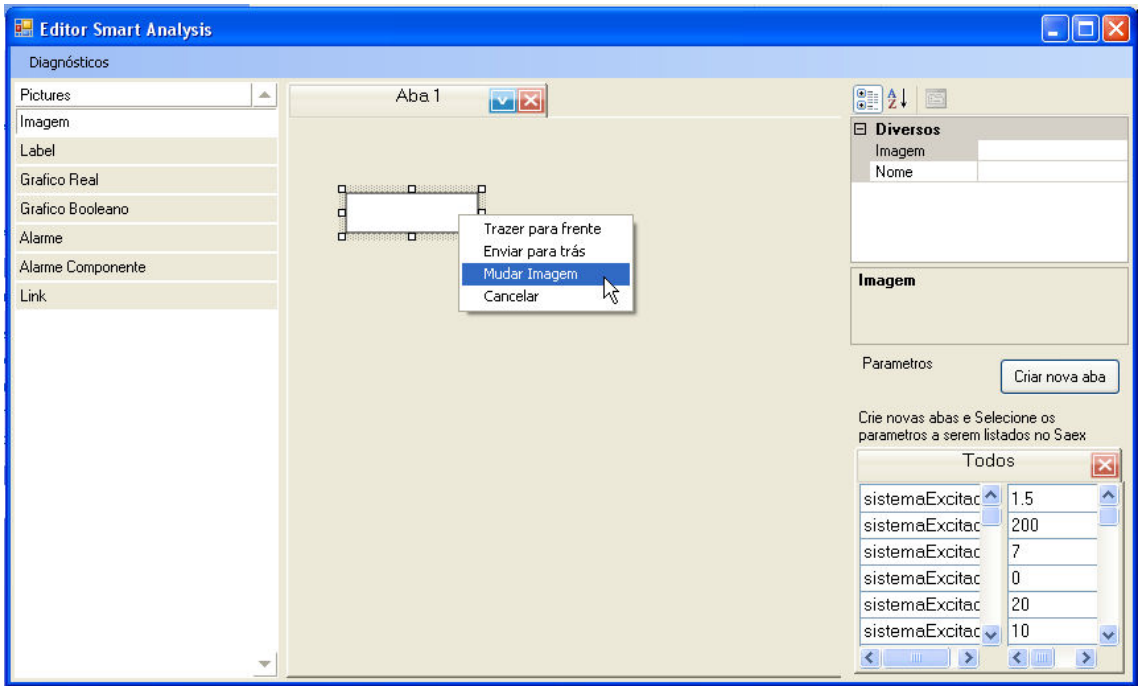


Figura 80: Auto-FDI Editor - Escolhendo uma imagem no editor

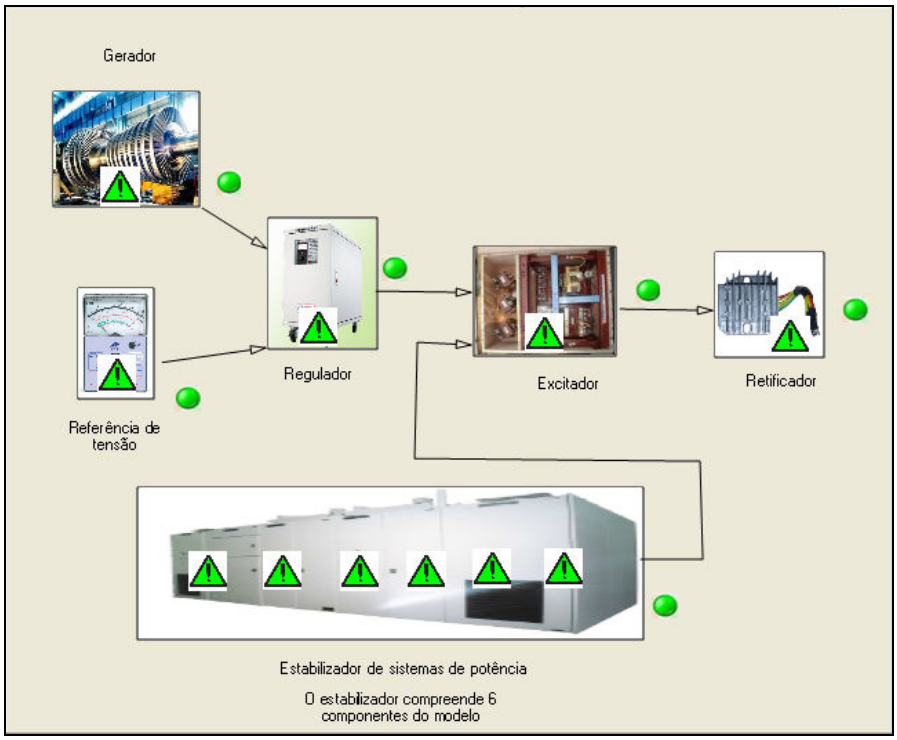


Figura 81: Representação visual do sistema

Vale notar que a representação visual que criamos para o sistema é diferente da estrutura do modelo do sistema criado no *SimForge* (veja na Figura 82). Isto é perfeitamente possível e muitas vezes desejável. Também é importante ressaltar que a visualização gráfica do sistema (imagens, links, labels) não tem nenhum significado durante a execução do diagnóstico.

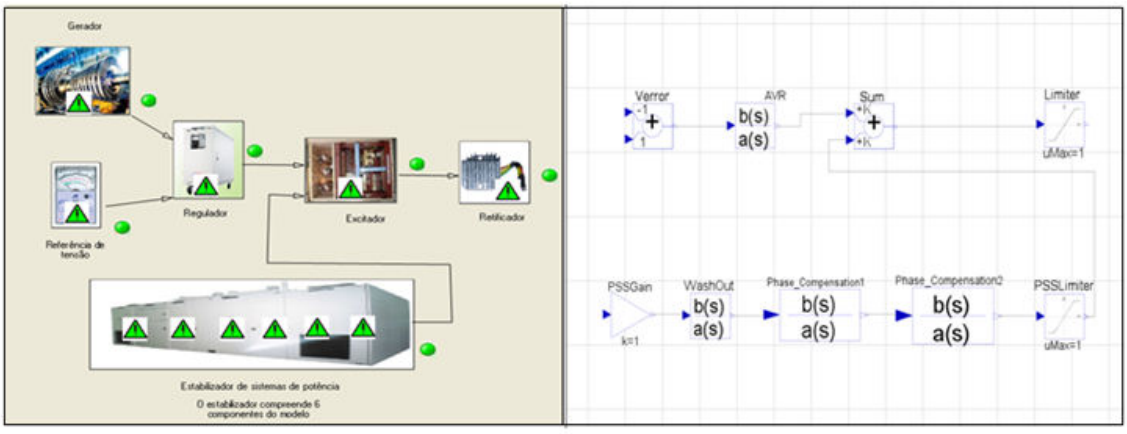


Figura 82: representação visual x estrutura do modelo

Observe agora os elementos em destaque na Figura 83. Estes são componentes de **alarme**.

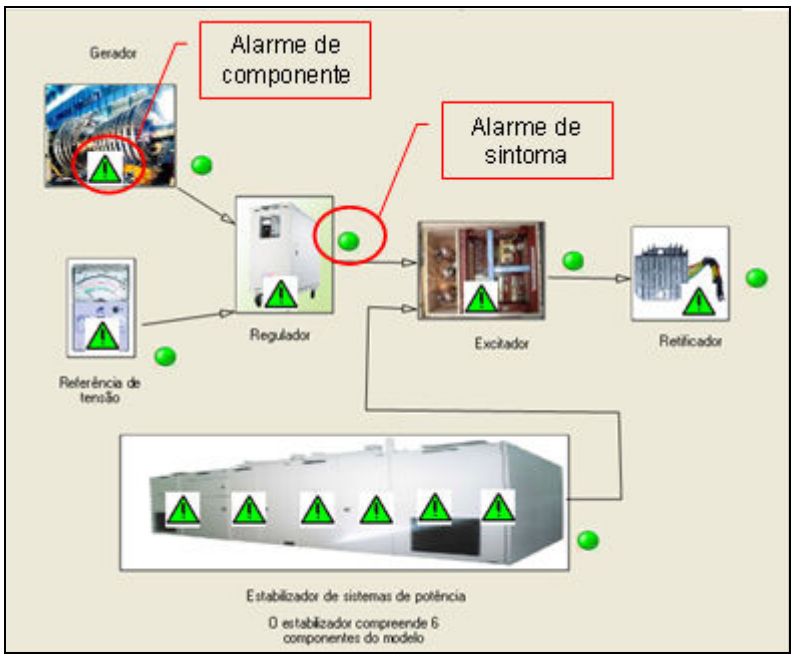


Figura 83: Alarmes

Os **alarmes de sintoma** (ou simplesmente alarme), como o próprio nome sugere, representam os sintomas gerados durante a detecção de falhas. Os sintomas são indícios da presença de falha no sistema. Durante a realização do diagnóstico, um alarme irá assumir a cor vermelha no intervalo de tempo em que componente do sistema ao qual ele está associado apresentar sintomas de falha.

Os **alarmes de componente** indicam que um determinado componente é uma fonte potencial de falhas. Os alarmes de componentes também assumem a cor vermelha na presença de falhas durante o diagnóstico.

A diferença entre alarmes de sintoma e alarmes de componentes pode ser evidenciada ao observar a Figura 84, obtida durante a execução do diagnóstico no *Auto-FDI Executor*. Como podemos observar na figura, somente o Gerador é realmente uma fonte potencial de falha (estado vermelho). Isso porque os sintomas de falha apresentados pelos outros equipamentos (Regulador, Excitador e Retificador) são possivelmente devido à propagação da falha do Gerador nestes componentes.

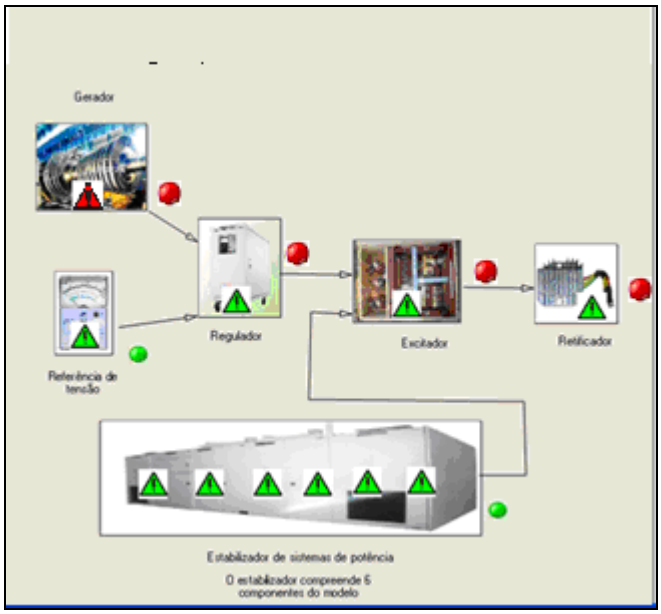


Figura 84: Alarmes de sintoma x alarmes de componentes

Ao arrastar um componente de alarme para a área de edição é exibida uma tela para a escolha do que será observado pelo alarme. No caso de um alarme de sintoma, são listadas todas as variáveis de saída do modelo de detecção de falha, que, de fato, são os sintomas. Para a alarme de componente é possível escolher entre todos os componentes do modelo do sistema. A Figura 85 ilustra a adição de um alarme de componente, com o componente "PSSGain" do modelo sendo observado pelo alarme.

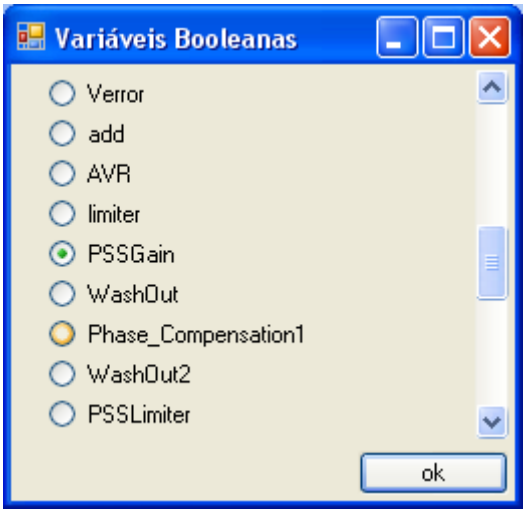


Figura 85: Adicionando um alarme de componente

É importante posicionar e associar os alarmes adequadamente de modo a permitir que, durante o diagnóstico, o operador possa associar facilmente o alarme ao equipamento ou sistema que apresenta a anomalia. Arraste os alarmes de modelo de

modo a obter algo equivalente ao mostrado na Figura 84, associando-os adequadamente aos componentes do modelo.

Vamos também adicionar gráficos para observar valores de variáveis durante a execução do diagnóstico. Iremos criar uma nova aba para adicionar os gráficos para não sobrecarregar a representação visual do sistema, embora fosse possível adicionar os gráficos na mesma aba.

Escolha a opção para criar uma nova aba na função no lado direito da aba existente e escolha a opção "Nova Aba". Também é possível alterar o nome das abas.

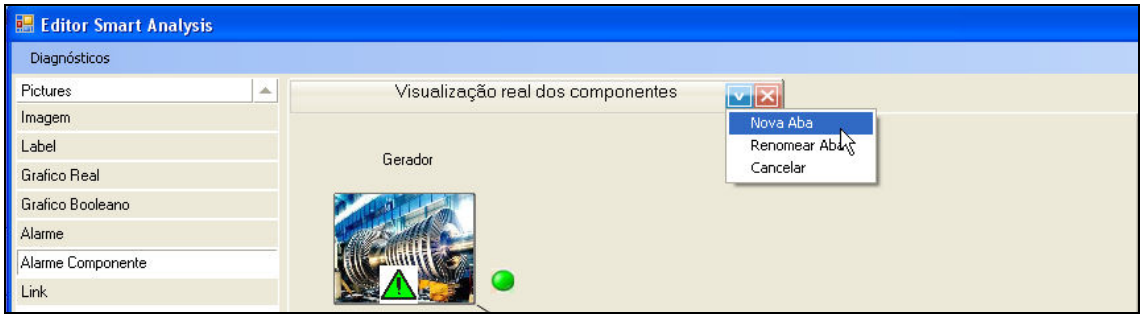


Figura 86: Auto-FDI Editor – Adicionando uma nova aba

Na nova aba, arraste um componente do tipo "Gráfico Real" e um do tipo "Gráfico Booleano". Ao adicionar um gráfico, será exibida uma tela para a escolha das variáveis a serem plotadas no gráfico, conforme mostra a Figura 87. A lista inclui as variáveis do modelo do sistema e do modelo de detecção.

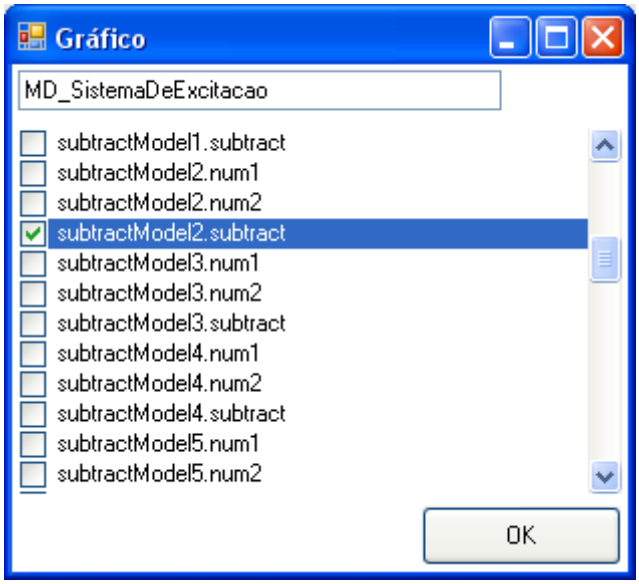


Figura 87: Escolhendo as variáveis a serem plotadas no gráfico

É possível posicionar e redimensionar os gráficos conforme desejado. A Figura 88 mostra os gráficos adicionados na aba.

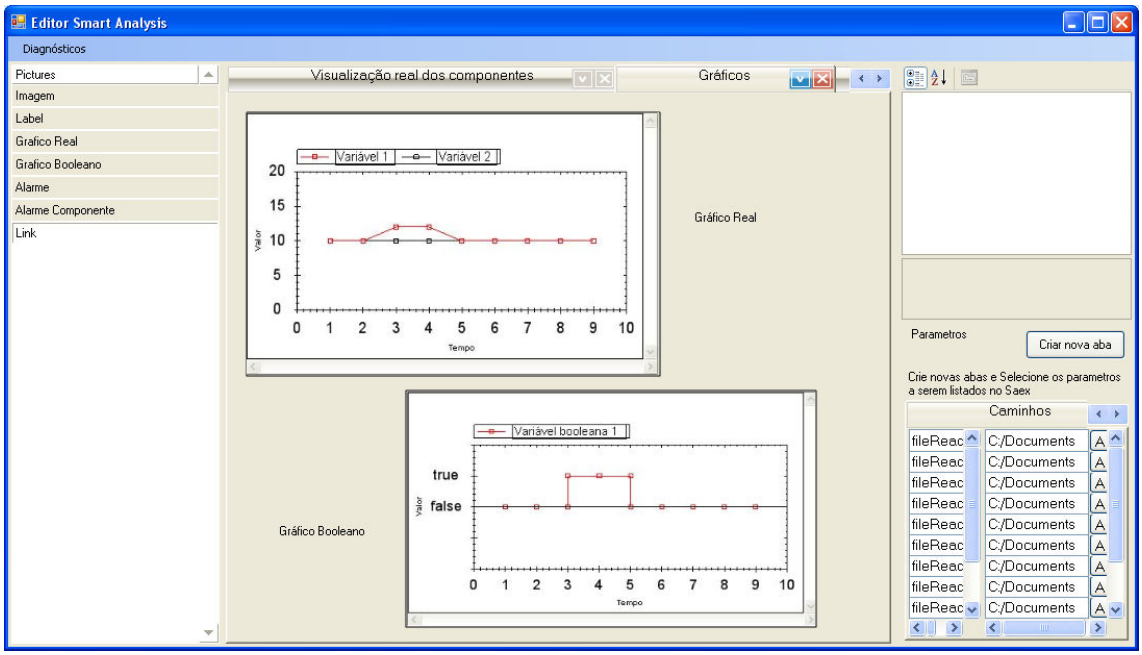


Figura 88: Auto-FDI Editor – Adição de gráficos

Em uma terceira aba, vamos colocar uma única imagem com a representação visual do modelo na linguagem Modelica. A imagem foi obtida através de um *printscreen* do modelo no *SimForge*. Em cima dessa imagem iremos colocar, em lugares apropriados, componentes do tipo "Alarme", conforme ilustra a Figura 89.

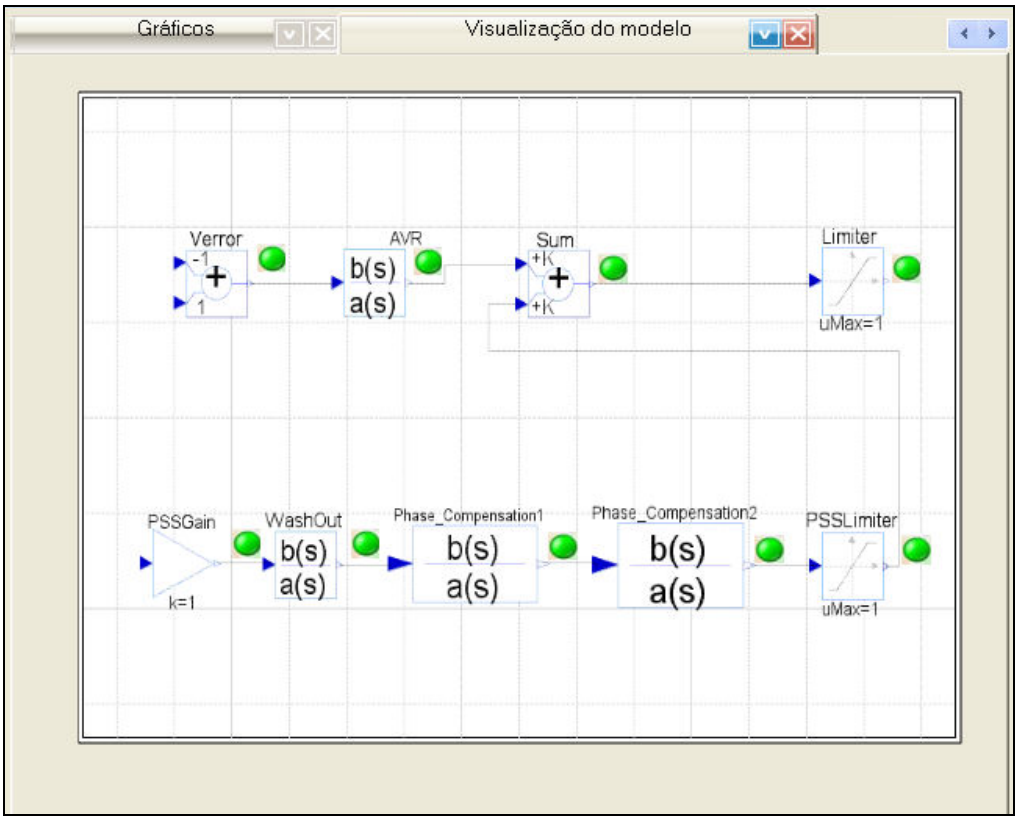


Figura 89: Auto-FDI Editor – Aba com imagem da estrutura do modelo

No lado direito do editor, em destaque na Figura 90, são listados todos os parâmetros do modelo do sistema e do modelo detecção de falha. O valor inicial dos parâmetros são aqueles que foram definidos durante a criação desses modelos no *SimForge*. Esses valores podem ser alterados.

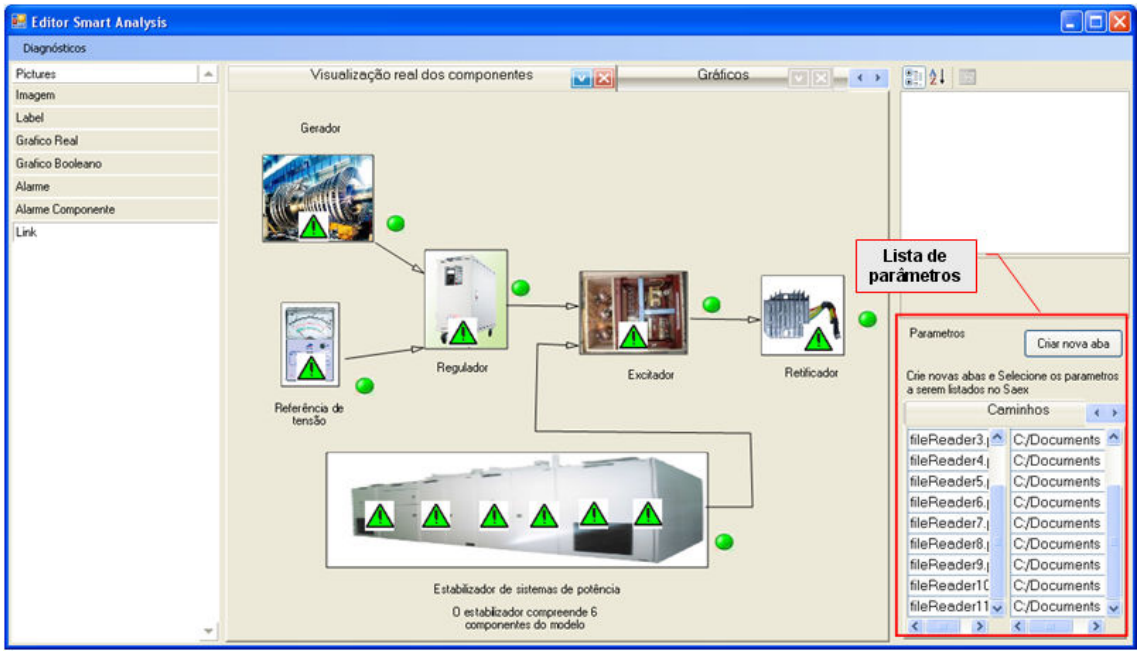


Figura 90: Auto-FDI Editor – Lista de parâmetros

É possível escolher quais os parâmetros poderão ser alterados no *Auto-FDI Executor* durante a preparação para realização do diagnóstico. Parâmetros que estiverem na aba "Todos" não poderão ser editados. Para que determinados parâmetros possam ser editados no *Auto-FDI Editor* é preciso colocá-los em outra aba de parâmetro.

Para criar uma nova aba clique em "criar nova aba" e escolha os parâmetros que serão adicionados na nova aba. Vamos selecionar os parâmetros dos avaliadores de resíduos, conforme ilustra o figura abaixo. Ao selecionar um parâmetro para a nova aba, ele será retirado da aba "Todos". Também é possível renomear a aba utilizando a função no lado direito da aba.

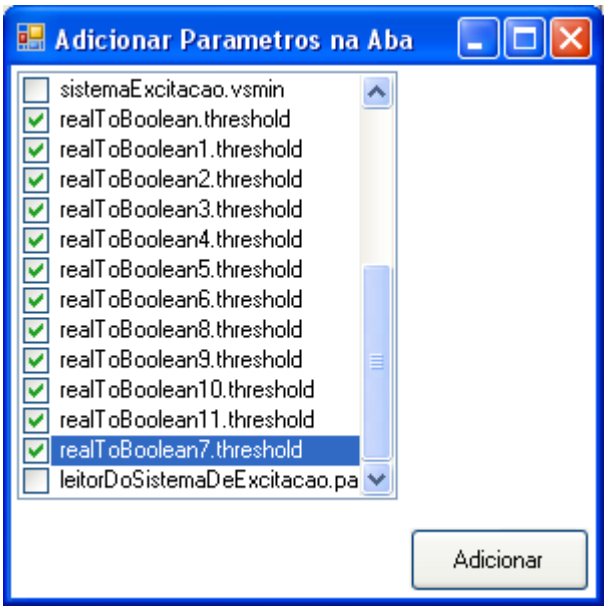


Figura 91: Auto-FDI Editor – Adicionando parâmetros em uma nova aba

Por fim, vamos salvar o ambiente criado. Para salvar utilize a opção "Salvar" ou "Salvar Como" no menu "Diagnósticos" e escolha o diretório onde a aplicação será salva. Na seção seguinte iremos usar o ambiente de diagnóstico criado para realizar o diagnóstico no sistema de excitação.

1.6 Realização do diagnóstico e avaliação dos resultados

Agora vamos utilizar o ambiente criado no *Auto-FDI Editor* para realizar o diagnóstico para o sistema de excitação. A realização do diagnóstico é feita no *Auto-FDI Executor*.

Para iniciar o *Auto-FDI Editor* clique na opção Smash Executor na tela inicial do *Auto-FDI*. O *Auto-FDI Executor* será aberto.

Agora vamos abrir o ambiente de execução que foi criado anteriormente no *Auto-FDI Editor*. Para isto, escolha a opção "Diagnósticos/Abrir" no menu e selecione o arquivo que foi salvo na etapa de montagem da ambiente. A figura abaixo ilustra o momento da escolha do arquivo.

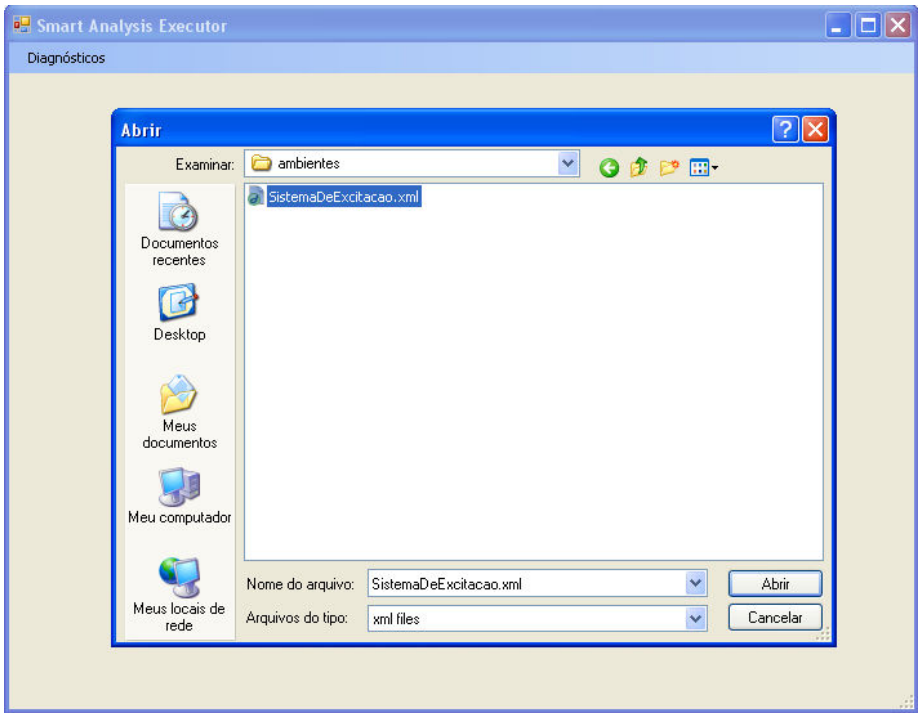


Figura 92: Escolhendo o ambiente de diagnóstico

Depois de abrir o arquivo será exibida a interface que criamos no *Auto-FDI Editor*. Observe os elementos da interface na figura abaixo.

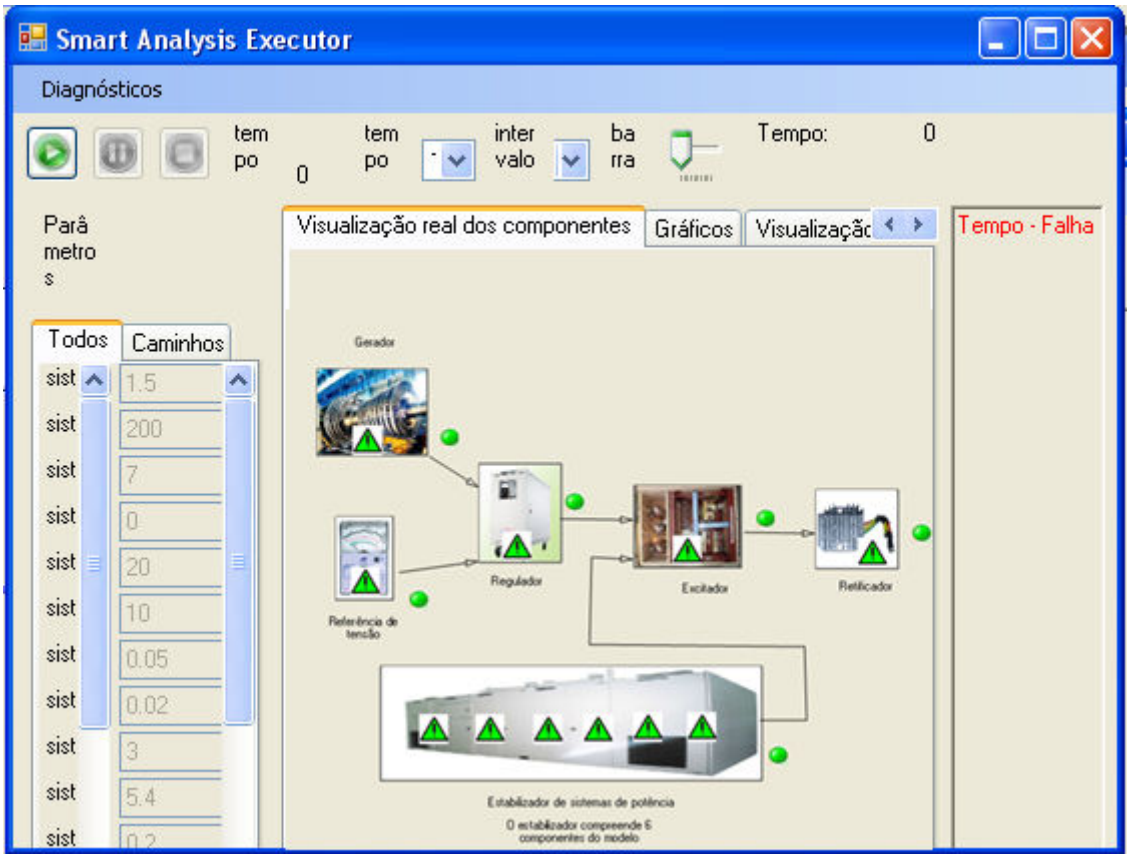


Figura 93: Auto-FDI Editor – Ambiente de diagnóstico aberto

No lado esquerdo da tela está a **janela de parâmetros**. Esta janela lista todos os parâmetros do modelo. Os parâmetros que não estiverem na aba "Todos" poderão ser alterados em cada execução do diagnóstico. Na parte superior estão as ferramentas de execução do diagnóstico. Através dessas ferramentas é possível iniciar, pause e parar a execução. Também existe opções para definir o tempo de simulação, o intervalo de verificação e controle da velocidade de simulação.

No lado direito da tela temos a **janela de exibição de falhas**. Nesta janela serão exibidos os componentes que apresentarem falha durante o diagnóstico.

Vamos agora dar inicio à execução do diagnóstico. Para isso, clique no botão iniciar execução na barra de ferramenta ( ). A Figura 94 ilustra a execução durante um determinado instante do diagnóstico. Observe que o *Auto-FDI* conseguiu identificar uma falha no sistema (no Gerador) e que as falhas se propagaram através dos componentes que estavam conectados ao Gerador.

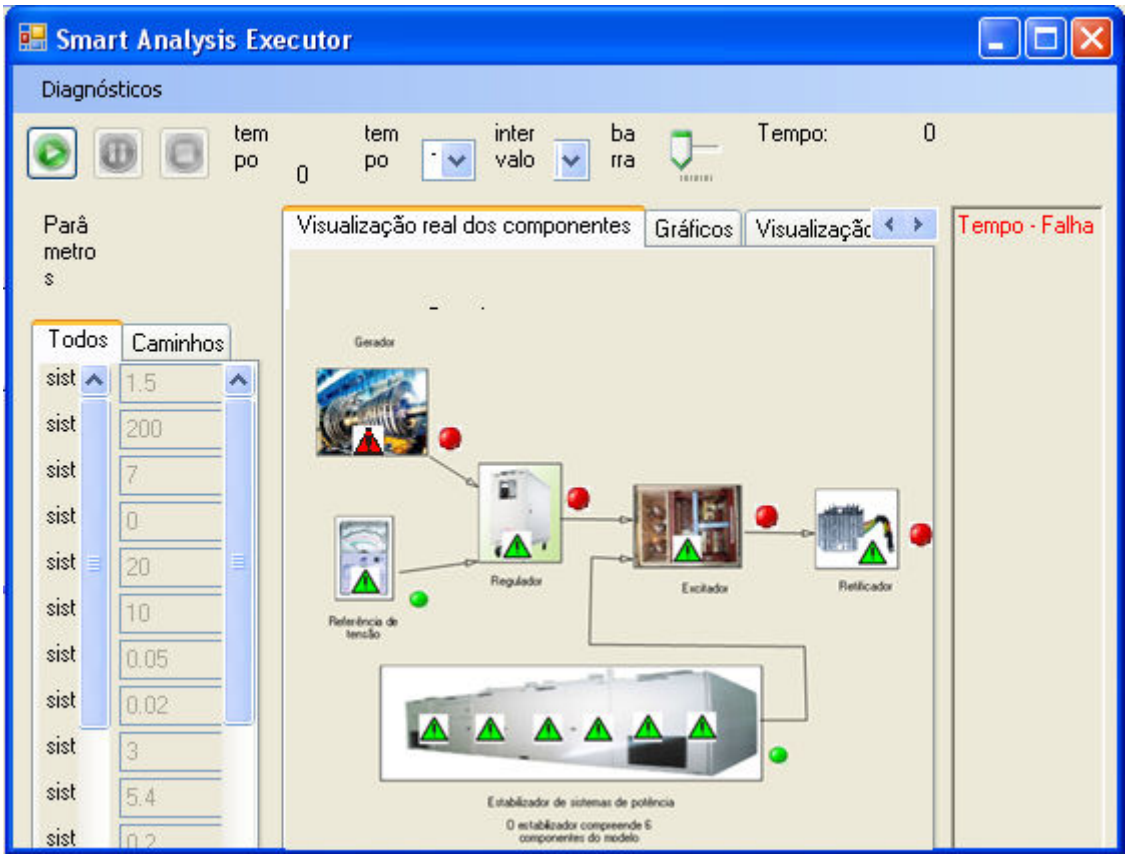


Figura 94: Execução do Diagnóstico - Identificação de uma falha no Gerador

1.7 Simulação do processo de detecção e diagnóstico de falhas

Conforme já mencionamos, o uso de adaptadores é o mecanismo do Auto-FDI para a injeção de sinais medidos de sistemas reais, permitindo o diagnóstico para tais sistemas. Alternativamente, o Auto-FDI pode ser usado para simulação do processo de detecção e diagnóstico de falhas, permitindo **injeção de falhas** no sistema avalidado.

Injeção de falhas é uma técnica bastante importante em sistemas de detecção e diagnóstico de falhas. Esta técnica consiste em inserir falhas no sistema e verificar se o resultado do diagnóstico em resposta à falha foi o esperado. No caso do Auto-FDI, as falhas são injetadas em um modelo que representa o sistema e não no sistema real, embora isso também seja possível. O mecanismo de injeção de falhas no Auto-FDI permite:

- Avaliar a qualidade do diagnosticador através de métricas de confiabilidade, sensibilidade a falhas, verificação da cobertura de falhas, etc.;

- Otimizar o processo de diagnóstico;
- Realizar treinamento dos operadores sem a necessidade de "conectar" ao sistema real;
- Entender melhor o modelo de detecção de falhas.

Para realizar a simulação do processo de detecção e diagnóstico de falhas é necessário, antes de tudo, ter um modelo de detecção que contenha um modelo do sistema com componentes especiais, chamados **componentes de injeção de falhas**.

Os componentes de injeção de falhas têm a finalidade de alterar valores das entradas e/ou saídas do modelo, simulando com isso as falhas. A biblioteca Diagnostics do Auto-FDI disponibiliza componentes para injeção tanto de falhas aditivas quanto falhas multiplicativas.

A Figura 95 ilustra um modelo de detecção de falhas criado para o sistema de excitação descrito neste documento. O Modelo de Detecção possui um modelo que descreve o comportamento ideal do sistema (sem falhas) e o modelo que representa o sistema real e que possui os componentes de injeção de falhas (modelo com falhas), além dos componentes que implementam a geração e avaliação de resíduos. Observe na figura que o modelo é praticamente idêntico ao modelo de detecção usado no diagnóstico real (veja Figura 75), diferenciando apenas pela substituição dos "adaptadores" pelo modelo com falhas.

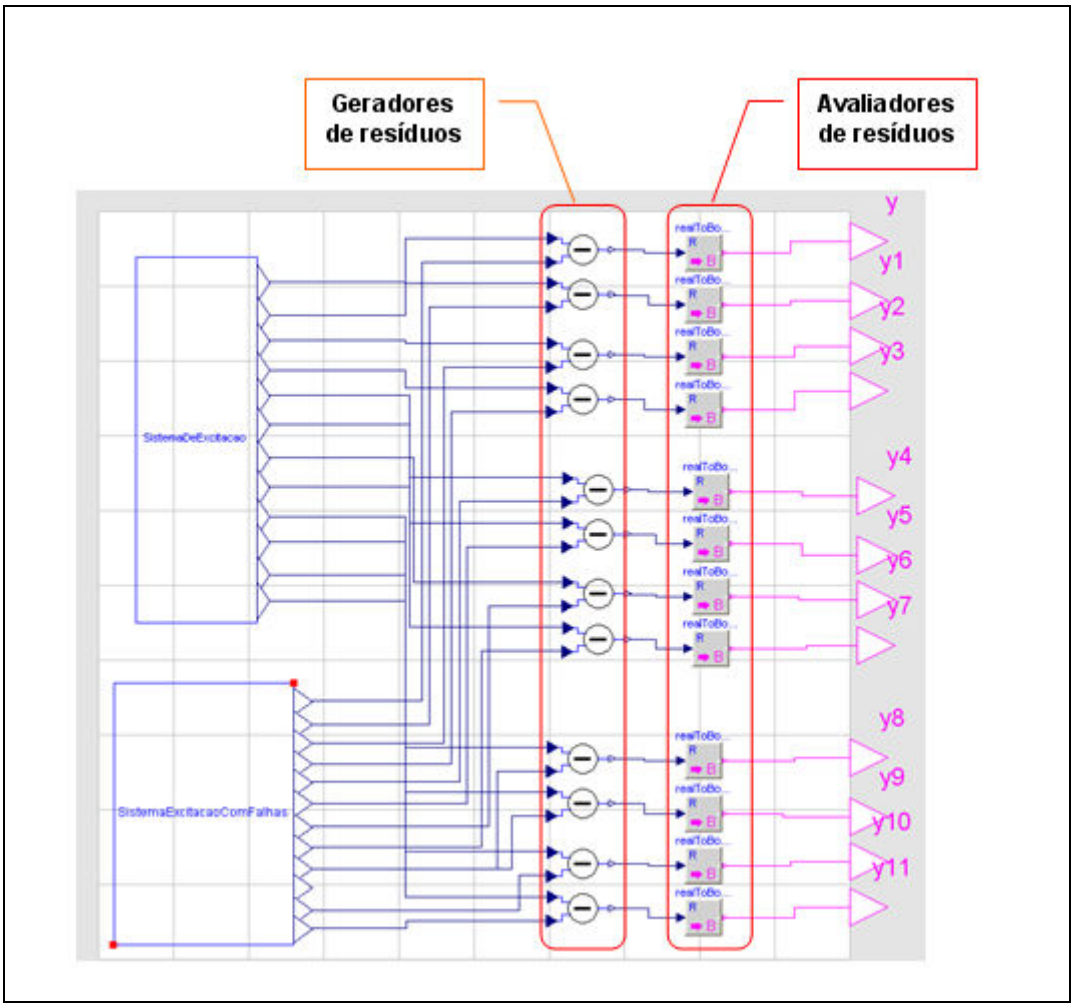


Figura 95: Modelo de Detecção para o Sistema de Excitação

A Figura 96 é a ilustração do modelo com falhas (SistemaExcitacaoComFalhas) usado no modelo de detecção da figura acima. Em destaque na figura estão os componentes de injeção de falha. Neste exemplo foram usados componentes de injeção de falhas aditivas (FalhaAditiva).

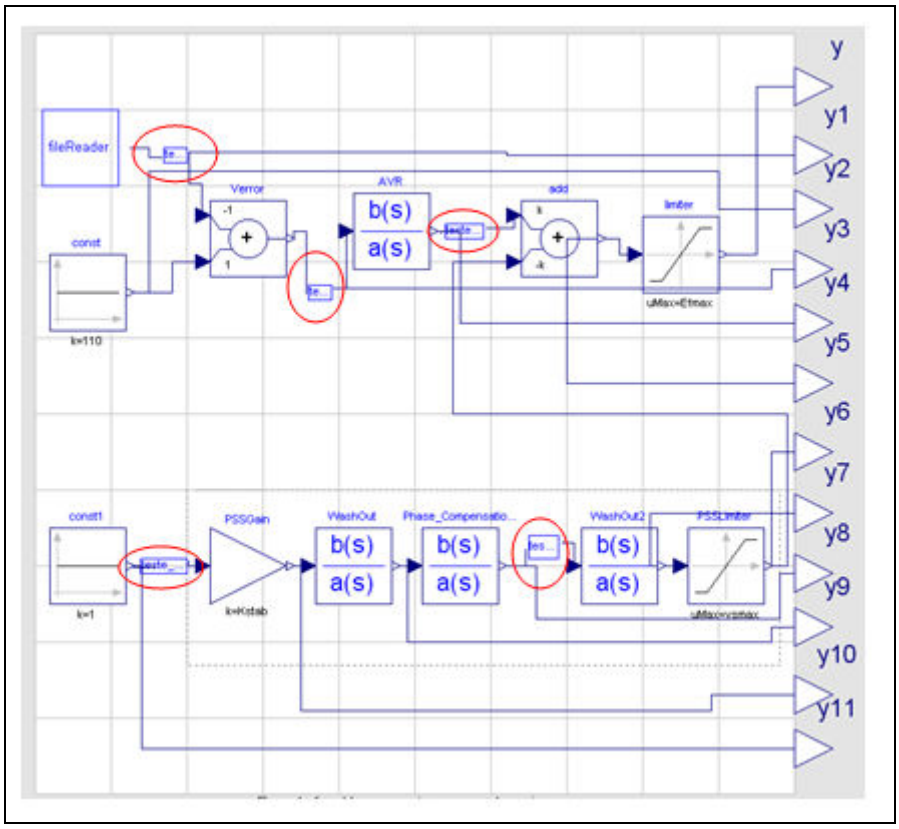


Figura 96: Modelo do Sistema de Excitação com componentes de injeção de falha

É importante salientar que os componentes de injeção de falhas devem ser inseridos no modelo interligados aos componentes aos quais se desejam injetar as falhas (na entrada ou saída de tais componentes). A Figura 97 ilustra como pode ser feito para permitir a injeção de falhas no componente "Phase_Compensation" do modelo. Observe que o componente de injeção de falha intercepta a saída do componente onde a falha poderá ser injetada.

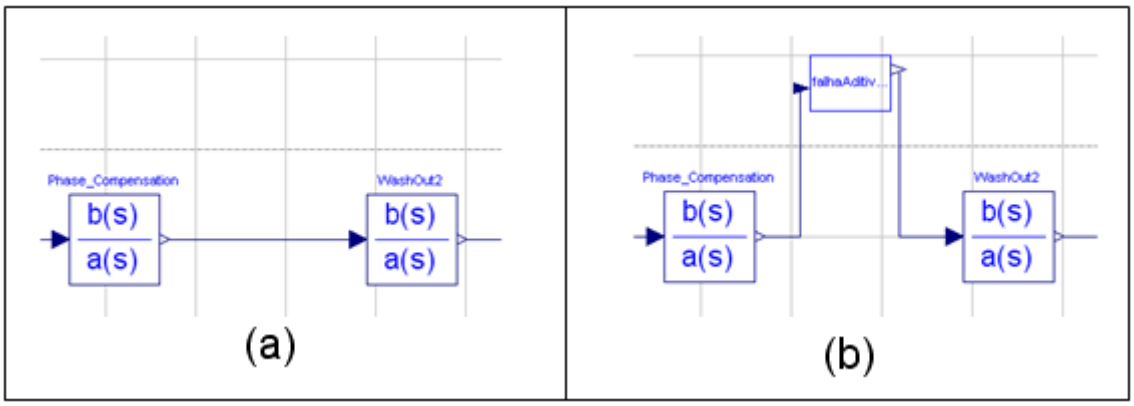


Figura 97: Injeção de Falhas: (a) modelo sem componente de injeção de falha; (b) modelo com componente de injeção de falha

Os componente FalhaAditiva usado no exemplo acima possui um parâmetro "valor". Esse parâmetro indica um valor real que será adicionado aos sinais de entrada/saída do modelo.

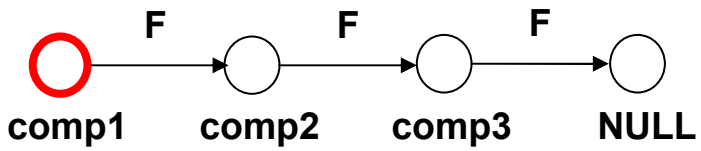
Os valores dos parâmetros dos componentes de injeção de falhas são alterados de mesma maneira que é feito em qualquer outro componente. Assim, pode ser feito no Dymola, no Auto-FDI Editor ou no Auto-FDI Executor conforme mostrado anteriormente.

A simulação do modelo de detecção criado pode ser feita diretamente no *SimForge*, permitindo a injeção de falhas e verificação de todas as variáveis do modelo através de gráficos. Alternativamente, pode ser criado o ambiente de simulação no Auto-FDI Editor, da mesma forma como é feito para um sistema real.

Apêndice D – Cenários de Isolamento Automático de Falhas

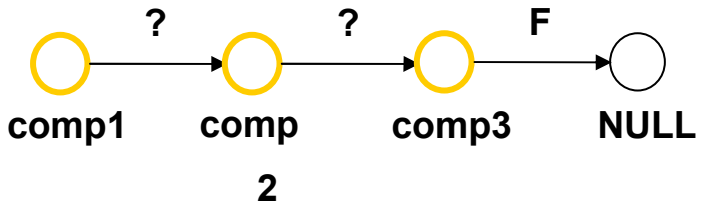
Neste apêndice apresentamos alguns cenários de modelo de propagação de falhas (FPM) com o respectivo resultado do isolamento automático. A cor vermelha representa que os respectivos componentes contêm falhas enquanto que a cor amarelha significa que o componente é um potencial fonte de falhas, mas não é possível determinar com certeza se o componentes apresenta falhas ou não.

FPM01



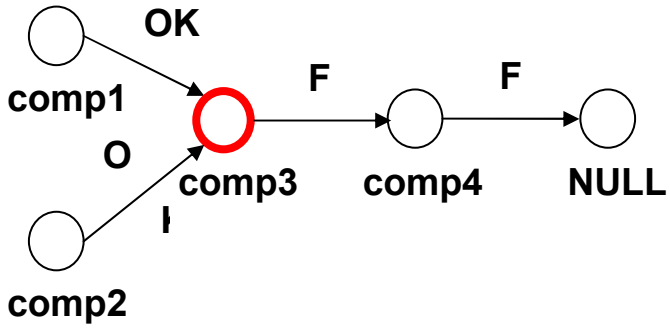
S = { {comp1} }

FPM02



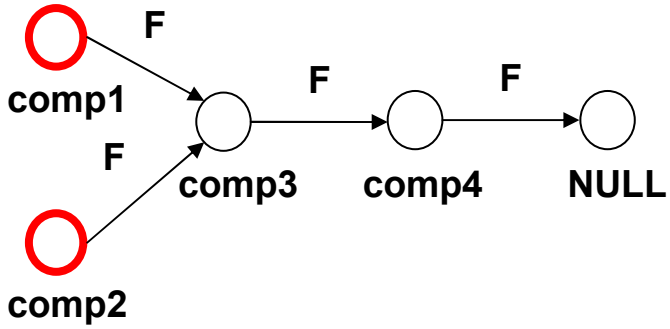
S = { {comp1, comp2, comp3} }

FPM03



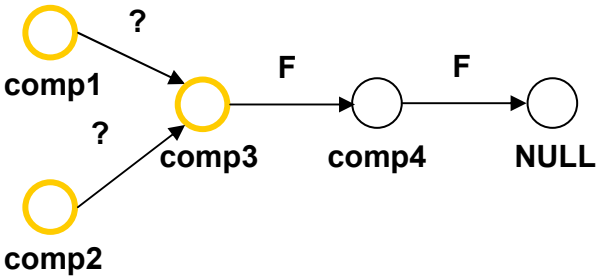
S = { {comp3} }

FPM04



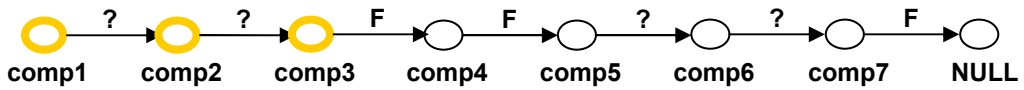
S = { {comp1}, {comp2} }

FPM05



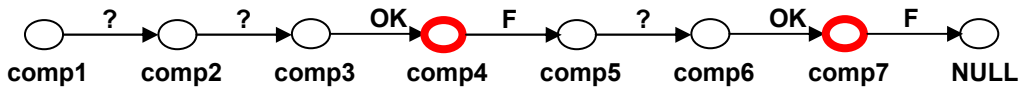
S = { {comp1, comp2, comp3} }

FPM06



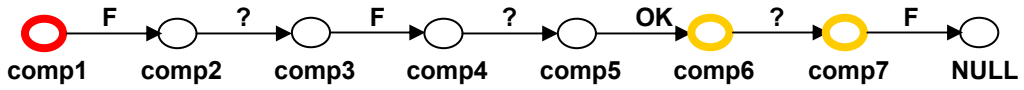
S = { {comp1, comp2, comp3} }

FPM07



S = { {comp4}, {comp7} }

FPM08

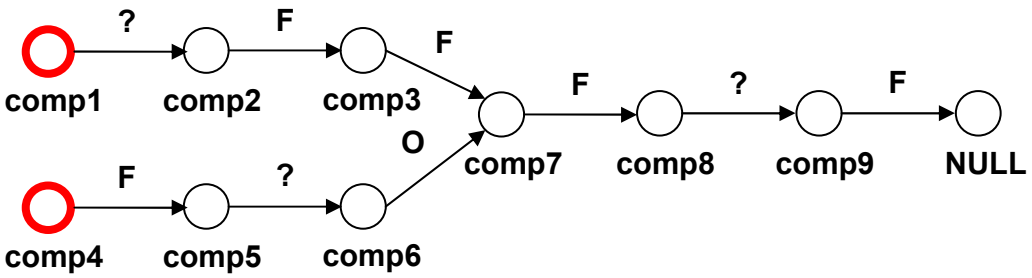


S = { {comp1}, {comp6, comp7} }

FPM08_2

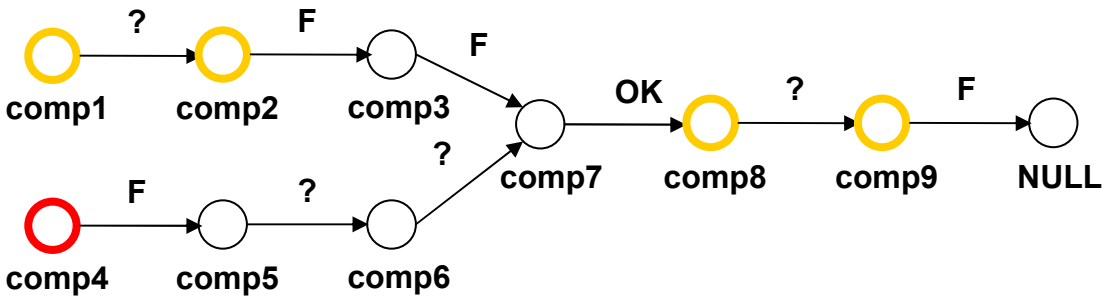
S = { {comp5, comp6, comp7} }

FPM09



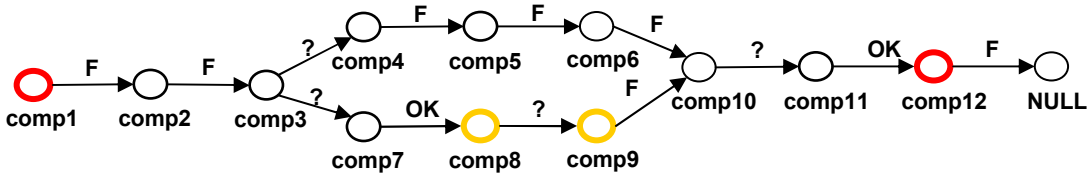
S = { {comp1, comp2}, {comp4} }

FPM10



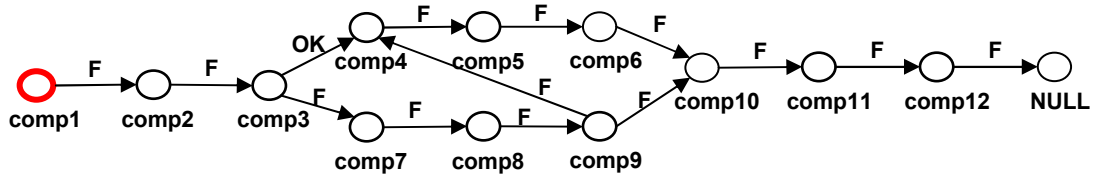
S = { {comp1, comp2}, {comp4}, {comp8, comp9} }

FPM11



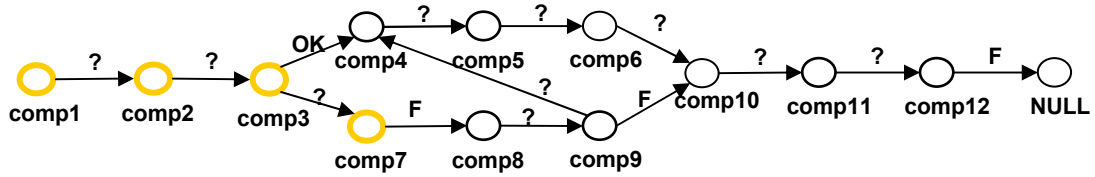
S = { {comp1 }, {comp8, comp9}, {comp12} }

FPM12



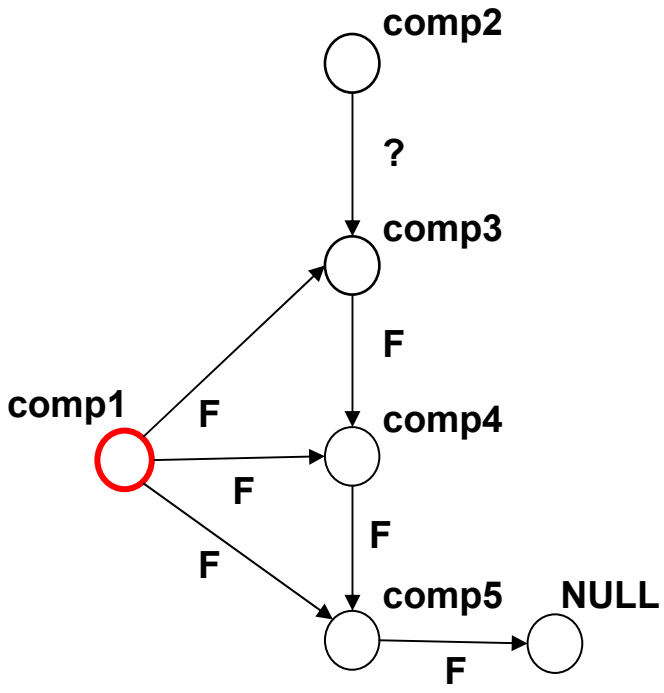
S = { {comp1} }

FPM13



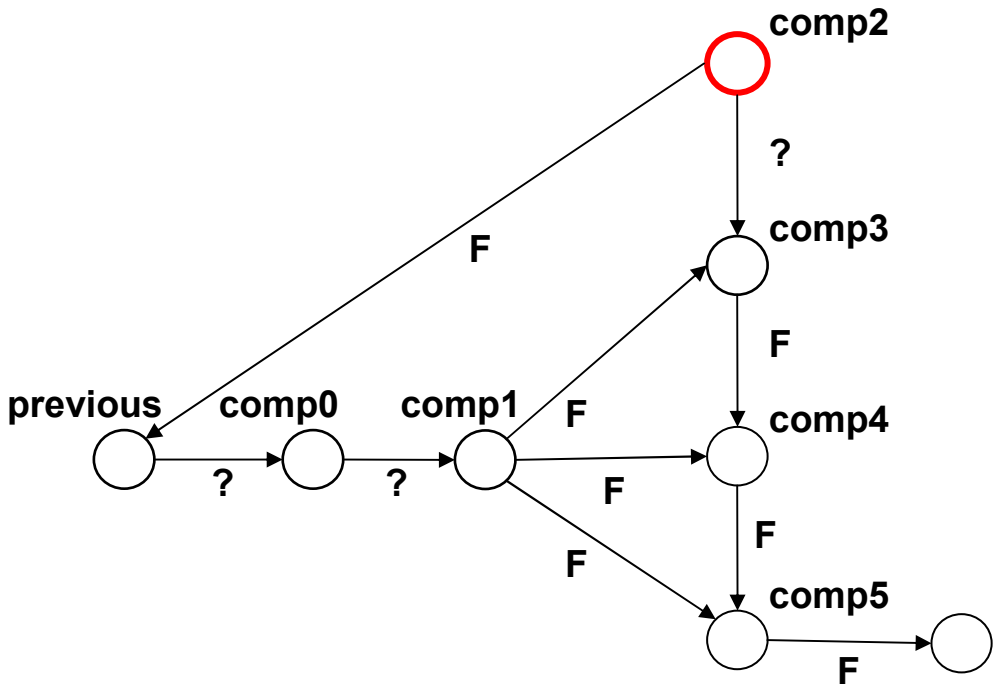
S = { {comp1, comp2, comp3, comp7} }

FPM14



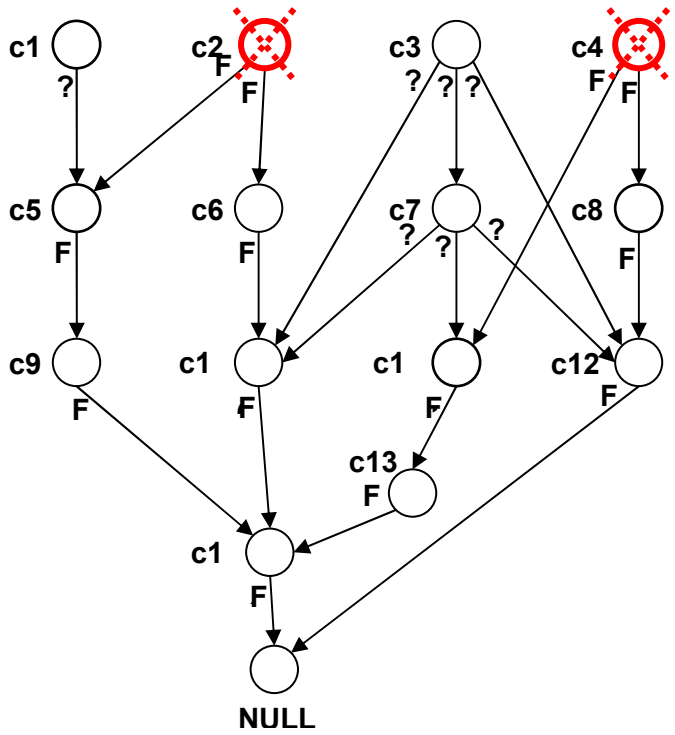
S = { {comp1 } }

FPM15



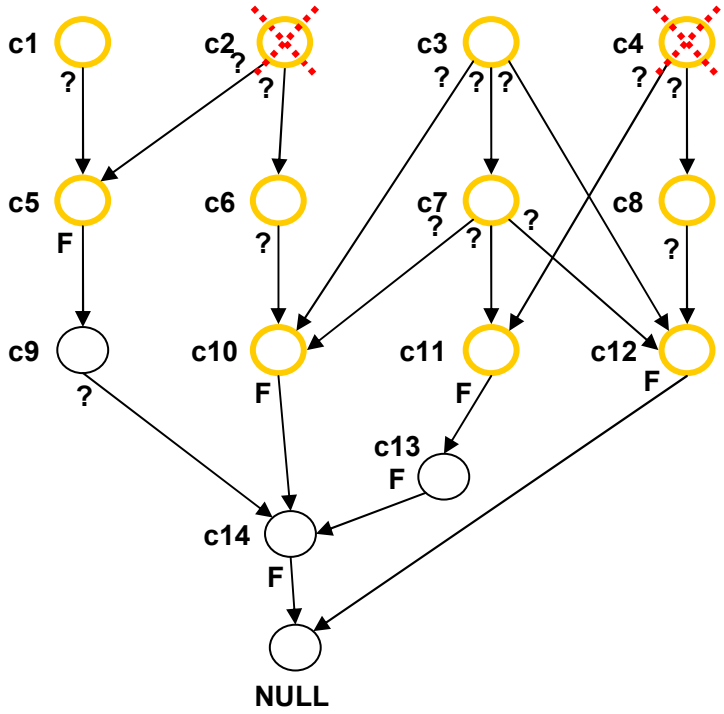
S = { {comp2 } }

FPM16 (adaptado de [CS2001])



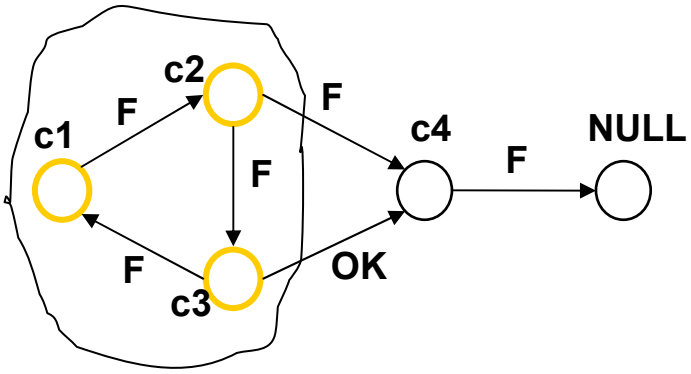
S = { {comp2 }, {comp4}}

FPM17 (from [Chessa e Santi])

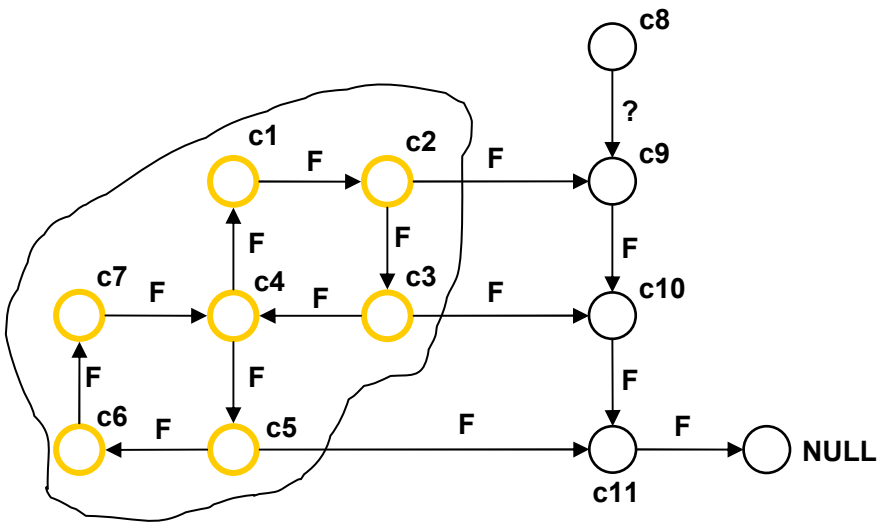


S = { {c1, c2, c5 }, {c2, c3, c6, c7, c10}, {c3, c4, c7, c11}, {c3, c4, c7, c8, c12 }}

FPM18



S = { {c1, c2, c3 } }



S = { {c1, c2, c3, c4, c5, c6, c7 } }

Bibliografia

[ASS2007] Abath Neto, O.L., Sauvé, J,; Souza, A. D. D., Patterns for Scripted Acceptance Test-Driven Development; In: Proceedings of the 12th European Conference on Pattern Languages of Programs, EuroPlop 2007, Irsee Monastery, Germany, 2007, pp. A4.1-A4.13.

[Beck1999] Beck, K., Extreme Programming Explained: Embrace Change, Addison-Wesley Professional, 1999.

[Beck2002] Beck, K., Test-Driven Development By Example, Addison Wesley, 2002

[BL2006] Blanke, M., Lorentzen, T., SaTool - A software tool for structural analysis of complex automation systems Fault Detection, Supervision and Safety of Technical Processes, IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2006.

[Bredemeyer-a] Bredemeyer Consulting, "Conceptual Architecture Action Guide", disponível em: http://www.bredemeyer.com/pdf_files/ActionGuides/ConceptualArchitectureActionGuide.PDF, acessado em maio de 2009.

[Bredemeyer-b] Bredemeyer Consulting, "Key Decisions Template", disponível em: http://www.bredemeyer.com/pdf_files/WhitePapers/KeyDecisionsTemplate.doc, acessado em maio de 2009

[BSS2006] Bouamama, B.O., Staroswiecki, M., Samantaray, A.K., Software for supervision system design in process engineering industry, IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2006.

[CS2001] Chessa, S.; Santi, P., "Operative diagnosis of graph-based systems

with multiple faults," Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on , vol.31, no.2, pp.112-119, Mar 2001

[CV1997] Cardoso, J. and Valette, R., Redes de Petri, Editora UFSC, 1997.

[DAS+2006] Ding, S.X., Atlas, E., Schneider, S., Ma, Y., Jeinsch, T., Ding, E.L., "An introduction to a Matlab-based FDI-toolbox Fault Detection, Supervision and Safety of Technical Processes", IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2006.

[DFS] Wikipedia. Depth-first search - Wikipedia, The Free Encyclopedia, acessado em maio de 2008.

[DPRS1995] Deb, S.; Pattipati, K.R.; Raghavan, V.; Shakeri, M.; Shrestha, R., "Multi-signal flow graphs: a novel approach for system testability analysis and fault diagnosis," Aerospace and Electronic Systems Magazine, IEEE , vol.10, no.5, pp.14-25, May 1995

[DV2000] S. Dash, V. Venkatasubramanian, "Challenges in the industrial applications of fault diagnostic systems", In: Process Systems Engineering (PSE 2000), Computers and Chemical Engineering, vol. 24, 2000, pp. 785-791.

[Dym] Dymola, site: http://www.dynasim.se

[ExtendedPetriNet] Extended Petri Net, site: http://www.modelica.org/libraries/ExtendedPetriNets

[FKNA2006] Frisk, E., Krysander, M., Nyberg, M., Aslund, J., A toolbox for design of diagnosis systems Fault Detection, Supervision and Safety of Technical Processes, IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2006.

[Fowler1997] Fowler, Martin; Scott, Kendall. UML Distilled, Addison-Wesley, 1997.

[FPDM2001] Fang Tu; Pattipati, K.R.; Deb, S.; Malepati, V.N.,

"Computationally efficient algorithms for multiple fault diagnosis in large graph-based systems," Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on , vol.33, no.1, pp. 73-85, Jan. 2003

[Freyumuth1991] Freyeumuth, B., "Knowledge-based incipient fault diagnosis of industrial robots," In Prepr. IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS), volume 2, 1991, pages 31-37.

[Fritzson2004] Fritzson, P., Principles of Object-oriented Modeling and Simulation with Modelica 2.1. IEEE Press, 2004

[Germoglio2008] Germoglio, G. Apoio para o ensino de projeto de arquitetura de software, Dissertação de Mestrado, COPIN/CCT/UFCG, 2008.

[Gertler1998] Gertler, Janos J., "Fault Detection and Diagnosis in Engineering Systems", Electrical Engineering & Electronics, 1998.

[GHJV1995] Gamma E.; Helm R.; Johnson R.; Vlissides J.; Design Patterns: Elements of Object-Oriented Software. Addison-Wesley, 1995.

[GSHC2006] Gros, R.; Sauter D.; Hamelin, F.; Corbier, F., "Fault monitoring & fault tolerant control in controlbuild software platform Fault Detection," IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2006.

[IB1997] Isermann, R.; Ballé, P., "Trends in the application of model-based fault detection and diagnosis in technical processes", Control Engineering Practice, Vol. 5, pp. 707-719, 1997.

[Isermann2004] Isermann, R., "Model-based fault-detection and diagnosis – status and applications", Annual Reviews in Control, 29, 2004, pp. 71-85.

[Isermann2006] Isermann, R., "Fault-Diagnosis Systems", Springer, 2006.

[IU1993] Isermann, R.; Ulieru, M., "Integrated fault detection and diagnosis," Systems, Man and Cybernetics, 1993. 'Systems

Engineering in the Service of Humans', Conference Proceedings., International Conference on , vol., no., pp.743-748 vol.1, 17-20 Oct 1993

[KKKC2004] Korbicz, J.; Kościelny, J.M.; Kowalczuk, Z.; Cholewa, W., "Fault Diagnosis: Models, Artificial Intelligence, Applications", Springer, 2004.

[KSW2006] Kościelny, J. M., Syfert, M., Wnuk, P., Advanced Monitoring and Diagnostic System 'AMandD', IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2006.

[Mod] Modelica and the Modelica Association, site: http://www.modelica.org

[ModelicaLibrary] Modelica Standard Library 3.0.1, site: http://www.modelica.org/libraries/Modelica

[Mono] Mono 2.4, site: http://mono-project.com

[Munoz1996] Muñoz San Roque, A., "Aplicación de Técnicas de Redes Neuronales Artificiales □o Diagnóstico de Procesos Industriales", Tesis doctoral. Escuela Técnica Superior de Ingenieros Industriales (ICAI). Universidad □ontifícia Comillas.

[NCover] NCover, site: http://www.ncover.com/

[NUnit] NUnit, site: http://www.nunit.org

[ObjectStab] ObjectStab 1.0, site: http://www.modelica.org/libraries/ObjectStab

[OMSD2004] B. Ould Bouamama, K Medjaher, A. K. Samantaray and G. Dauphin-Tanguy. Model builder using functional and bond graph tools for FDI design, Control Engineering Practice (CEP) journal 13(7), 875-891. 2004

[OpenMod] OpenModelica, site: http://www.ida.liu.se/~pelab/modelica/OpenModelica.html

[Pontes2008] Pontes, Renata França. Um Framework Para Auxílio Na Criação

De Diagnosticadores De Falhas Em Sistemas, Dissertação de Mestrado, COPIN/ CCT/UFCG, Fevereiro, 2008.

[Preiss98] Preiss, Bruno R.; Data Structures and Algorithms with Object-Oriented Design Patterns in C++, Wiley, 1998.

[Rao1993] Rao, N. S. "Computational Complexity Issues in Operative Diagnostics of Graph-Based Systems," IEEE Trans. Comput. 42, 4 (Apr. 1993), 447-457

[Rao1996] Rao, N.S.V., "On parallel algorithms for single-fault diagnosis in fault propagation graph systems," Parallel and Distributed Systems, IEEE Transactions on , vol.7, no.12, pp.1217-1223, Dec 1996

[Sauve2007] Sauvé, J. "Projeto Arquitetural do Sistema SIEx, Projeto SIEx", 2007.

[SFP2002] Simani, S.; Fantuzzi, C.; Patton, Ron J., "Model-based Fault Diagnosis in Dynamic System Using Identification Techniques", Springer, 2002.

[Sim] SimForge, site: https:trac.elet.polimi.it/simforge

[Sommerville2004] Sommerville, Ian, Software Engineering, 7th Edition, Addison-Wesley, 2004

[SS2004] Steinder, M.; Sethi, A. S., "A survey of fault localization techiques in computer networks," Science of Computer Programming, Special Edition on Topics in System Administration, Vol. 53, pp. 165-194.

[Tarjan1972] Tarjan, R.E., "Depth-First Search and Linear Graph Algorithms," SIAM J. Computing, vol. 1, no. 2, pp. 146-160,1972.

[TBR+1999] Tesar, L., Berec, L., Rakar, A., Szederkényi, G., Kadlec, J., Juricic, D., Hangos, K.M., Kárny, M. e Kinnaert, M.: "A toolbox for model-based fault detection and isolation", Proceedings of the European Control Conference, 1999.

[TBR+1999] Tesar, L., Berec, L., Rakar, A., Szederkényi, G., Kadlec, J., Juricic, D., Hangos, K.M., Kárny, M. e Kinnaert, M.: A toolbox for model-based fault detection and isolation. Proceedings of the European Control Conference, 1999.

[WK2005] Wang, A.; Qian, K., Component-Oriented Programming. Wiley-Interscience, 2005